MASTER IN COMPUTER SCIENCE

Feature Model Extraction from Large Collections of Informal Product Descriptions

Davril, Jean-Marc; Delfosse, Edouard

Award date:
2013

Awarding institution:
University of Namur

Abstract

Feature Models (FMs) have become one of the most popular form of representation to model the commonalities and the variabilities of products in Software Product Lines (SPL) engineering. The task of manually creating an FM can be tedious and error-prone, which is why researchers have developed techniques for automatically extracting FMs from formally defined product descriptions (artifacts). The problem is that those artifacts are often not available, which is why we present a novel, fully automated approach for extracting FMs from publicly available product descriptions that can be found on software repository websites such as SoftPedia and CNET. While each individual product description provides only a partial view of features in the domain, a large set of descriptions can provide a fairly comprehensive coverage. Our approach utilizes hundreds of product descriptions to construct an FM and is described and evaluated against antivirus product descriptions mined from SoftPedia.

Les FMs sont devenus une des formes les plus populaires pour représenter les points communs et les points de variabilité des produits dans l'ingénierie des Lignes de Produits Logiciels. Créer manuellement un FM est une tâche fastidieuse et sujette à erreurs. C'est pourquoi les chercheurs ont développé des techniques pour extraire automatiquement des FMs depuis des descriptions de produits décrites formellement (artefacts). Le problème est que ces artefacts ne sont pas disponibles la plupart du temps. C'est pourquoi nous présentons une approche nouvelle, complètement automatisée pour extraire des FMs depuis des descriptions de produits accessibles à tout le monde, qui peuvent être trouvées sur des référentiels de logiciels tels SoftPedia et CNET. Alors que chaque description de produit individuelle ne fournit qu'une vue partielle des fonctionnalités du domaine, un grand ensemble de descriptions peut fournir une couverture assez complète. Notre approche utilise des centaines de descriptions de produits afin de construire un FM et elle est expliquée et évaluée avec des descriptions de produits antivirus extraits depuis SoftPedia.

Acknowledgements

The work described in this thesis is the result of a five months internship at DePaul University in Chicago, United States. We were part of the Requirements Engineering team, where we had the help of Negar Hariri (PhD student), and were supervised by Dr. Jane Cleland-Huang.
As a result of this internship, we wrote a paper called "Feature Model Extraction from Large Collections of Informal Product Descriptions" which was published at the main conference of ESEC/FSE'13[1], and we presented this paper on the 22^{nd} of August 2013 in Saint-Petersburg, Russia.

We would like to thank Dr. Patrick Heymans for giving us the opportunity of doing this internship, and for his supervision throughout. We would also like to thank Negar Hariri and Mathieu Acher for their precious help throughout the entire internship as well as during the writing of the paper. And last but not least, we would like to thank Dr. Jane Cleland-Huang for her supervision during the internship, her help in orientating us in the research process and in writing the paper. We would also like to thank her and her team at DePaul University for making our internship in Chicago such a great experience.

[1] http://esec-fse.inf.ethz.ch/

Contents

List of Figures

List of Tables

Glossary

Frequent Pattern Growth	Algorithm introduced in [HPY00] for mining *frequent itemsets*.
Text-based Variability Language	TVL is a text-based language used to represent FMs http://www.info.fundp.ac.be/~acs/tvl/.
CFPGrowth	Algorithm introduced in [LHM99] for mining association rules with multiple minimum supports.
feature descriptor	*Raw feature descriptor* after preprocessing.
frequent itemsets	Sets of terms that frequently co-occur together in a given context.
raw feature descriptor	Description of a feature in a descriptive paragraph or in a bulleted list.
transaction	A transaction in association rules mining or frequent itemset mining is a set of items that occurred together (e.g., {milk,break} would be a possible transaction in a supermarket.

Acronyms

CTC	Cross-Tree-Constraint.
DAG	Directed Acyclic Graph.
FD	Feature Diagram.
FG	Feature Graph.
FM	Feature Model.
FODA	Feature-oriented Domain Analysis.
FP-Tree	Frequent-pattern tree.
FPGrowth	Frequent Pattern Growth.
GUI	Graphical User Interface.
IG	Implication Graph.
PFM	Probabilistic Feature Model.
POS	Part-of-Speech.
SCC	Strongly Connected Component.
SPKMeans	Spherical k-means.
SPL	Software Product Lines.
tf-idf	Term frequency-inverse document frequency.
TVL	Text-based Variability Language.

Chapter 1

Introduction

This thesis studies the extraction of Feature Models (FMs) from large collections of textual descriptions of products. We propose an automated FM extraction procedure that aims to offer support for domain engineering tasks that are related to Software Product Liness (SPLs).
This chapter introduces the research problem studied throughout the thesis, the approach to this problem, the contribution of the thesis and the structure of the document.

1.1 Problem Description

SPL engineering usually covers two distinct phases of domain engineering and application engineering [WL99]. Domain engineering involves analyzing a specific domain, discovering commonalities and variabilities, and then constructing core assets which will be used across the entire SPL [SvGB05]. In contrast, application engineering is concerned with building a specific product based upon the core assets of the product line. In this thesis we focus on the process of constructing an FM as part of the domain engineering process. This process can be exceedingly time-consuming, and yet it can provide support for the domain engineering phase of the SPL engineering process (e.g., in performing assisted product configurations, designing a new family of products, expanding an existing product line, or simply inspiring the requirements elicitation phase of a single application process) and it can also help reduce the time-to-market.

We address the specific problem of building FMs from textual product descriptions. This is a tedious and error-prone task and the SPL research community has shown significant interest in the ability of automatically generating FMs from existing data. While existing approaches usually assume that products are formally and exhaustively defined as configurations of features, we are interested in mining FMs from informal and incomplete

product descriptions. We focus on the application of an FM synthesis procedure on software repository websites which publicly provide a large amount of textual descriptions for various types of software products. By targeting publicly available descriptions, we want to provide domain engineering support not only to organizations that do not have formal representation for their collections of products but also to organizations that have not previously developed products for a targeted domain.

It seems too ambitious to look for an automated approach to analyze textual and incomplete product descriptions that provides FMs containing the precise and exhaustive information to configure products. Our intent is rather to study the design of an FM extraction procedure that produces FMs which consist of an approximation of a product line representation and which offer a clear and useful overview of the targeted domain. Our goal is to provide support for domain analysis tasks and to ease the task of manually constructing FMs for practitioners.

The thesis describes the various steps of the FM extraction procedure and evaluates the quality of the extracted FM through a quantitative approach.

1.2 Approach

There are several examples of prior work in the area of FM synthesis. Czarnecki et al. [CSW08] introduced Probabilistic Feature Models (PFMs) and provided an extraction procedure that mined propositional formulas and soft constraints from a set of multiple configurations. However, their approach assumes that products are already formally described as sets of features. Similarly, Acher et al. [ACP+12] describe a way of extracting FMs from product descriptions, but their approach assumes the availability of formal and complete descriptions of products as configurations of features. Chen et al. and Weston et al. described techniques for extracting an FM from informal specifications ([CZZM05], [WCR09]). This approach is particularly useful in cases where an organization has an existing set of individual products and wishes to move towards an SPL approach. However, it also has specific limitations, because this approach assumes the completeness of the descriptions and makes assumptions regarding the grammatical patterns and the lexicon used in the text to describe the information regarding the variabilities between the requirements and features. Usually, existing approaches to extract FMs from data also require some manual intervention from the user to guide the inside hierarchical structure of the FM.

In this thesis we focus on analyzing publicly available data from websites such as SoftPedia[1], CNET[2], and MajorGeeks[3], which provide descriptions and feature lists for hundreds of thousands of products [DGH+11]. Product descriptions available on such sites are gen-

[1] http://www.softpedia.com/
[2] http://download.cnet.com/windows/
[3] http://majorgeeks.com/

erally incomplete, and features are described informally using natural language, making the existing approaches unsuitable. Our contribution is the proposal of an automated procedure to extract FMs from such descriptions. The procedure involves a combination of data mining and text-mining techniques applied to the collection of product descriptions. The key motivation behind our approach is the assumption that while each individual product description provides only a partial view of features in the domain, a large set of descriptions can provide a fairly comprehensive coverage. The development of some of the techniques applied during the procedure has been inspired by the way practitioners build FMs. Thus, the thesis proposes an analysis of automated solutions in attempt to replicate some of their practices.

The contribution of this work is the proposition and the analysis of an automated process that aims to efficiently capture information related to variabilities and similarities between products from textual descriptions that are publicly available on software repository websites, as well as the presentation of this information to users through the generation of FMs.

1.3 Structure

This thesis is structured in the following way: Part I presents background information about SPLs and FMs in particular. It also introduces basic concepts of data mining, domain analysis and concept mining that will later be used in the process we designed. In Chapter 3 we present a state of the art for FM mining techniques.

Part II is dedicated to presenting our extraction process. Chapter 4 presents an overview of the process. In Chapters 5, 6 and 7 we describe the different steps (mining of features, construction of a Implication Graph (IG) and structuring of the FM). Chapter 8 presents the tool that we have developed.

In Part III we present the results of the evaluation we conducted, which suggest that our extraction process is better than using an approach only based on statistical analysis. Finally we summarize our contribution, present a critical outlook and then propose suggestions for future work.

Part I

Background and Related Work

Chapter 2

Background

In this chapter we provide background information about the different concepts and methods that we use throughout our process. We first present the concept of SPLs in Section 2.1. Then we explain the domain analysis phase of SPL engineering in Section 2.2 and in Section 2.3 we explain the concept of FMs. Finally, we move on to data mining techniques in Section 2.4.

2.1 Software Product Lines

Nowadays, software companies need to have shorter time-to-market and low production costs in order to be competitive. This can be achieved using SPLs. According to the Software Engineering Institute, SPLs *"epitomize strategic, planned reuse, and represent a way of doing business that results in order-of-magnitude improvements in cost, time-to-market, and productivity''* [Ins, SV02]. They define an SPL as "*a set of software-intensive systems that share a common, managed set of features developed from a common set of core assets in a prescribed way*" [CN01, PBvdL05].

To get a better understanding of the usefulness of SPL engineering, we will use the case of a phone manufacturer who gives its clients the option to configure their phone with different features in Section 2.3.

2.2 Domain Analysis

Neighbors [Nei80], who can be considered as the founder of domain analysis, wrote in 1980 that *"Since domain analysis describes a collection of possible systems, it is difficult to create a good domain analysis. If only one system is to be built, then classical systems analysis should be used. A domain analysis is only useful if many similar systems are to be built so that the cost of the domain analysis can be amortized over all the systems"* [Nei80]. We can clearly see in those sentences that domain analysis applies well to SPL engineering, as the purpose is to build different systems. Neighbors also wrote that *"the key to reusable software is captured in domain analysis in that it stresses the reusability of analysis and design, not code"* [Nei80].

The purpose of the domain analysis phase in SPL engineering is to discover commonalities and variabilities, and then construct core assets which will be used across the entire SPL [SvGB05]. There are several methods that can be used for domain analysis: *Domain Analysis and Reuse Environnment (DARE)* [FPDF98], *Feature-oriented Domain Analysis (FODA)* [Ins] or *Integration Definition for Function Modeling (IDEF0)* [oST93]. The method we will focus on is FODA because it was intentionally created for SPL engineering. Also, as Czarnecki and Eisenecker wrote in [CE00], FMs are *"the greatest contribution of domain engineering to software engineering"*. IDEF0 is more popular for creating Domain Specific Modeling Languages [IFFD10].

In the FODA feasibility study done by the Software Engineering Institute [KCH+90], they pointed out different phases of the domain analysis and their corresponding products: Context analysis, Domain modeling and Architecture modeling. Those phases are depicted in Figure 2.1.

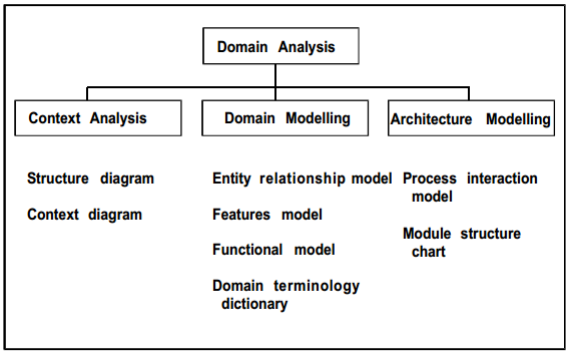


Figure 2.1: Phases and Products of Domain Analysis [KCH+90]

2.3 Feature Model

FMs are one of the most popular formalisms for modeling and reasoning about commonality and variability of an SPL [CGR+12]. Depending on the level of abstraction, and artifacts described, features may refer to a prominent or distinctive user-visible characteristic of a product or to an increment in a software code base [AK09, AB11, CHS08a]. A recent survey of variability modeling showed that FMs are by far the most frequently reported notation in industry [BRN+13]. Several academic or industrial tools have been developed to specify them graphically or textually and automate their analysis, configuration or transformation [pur, Big, BSRC10, ACLF13, TKB+12, CBH11].

FMs hierarchically organize a potentially large number of concepts (features) into multiple levels of increasing detail, typically using a tree. Features can be seen as "anything users or client programs might want to control about a concept", "a prominent or disjunctive user-visible aspect, quality or characteristic of a software system or systems" to "an increment in product functionality" [CHS08b]. Variability is expressed in terms of mandatory, optional and exclusive features as well as logical constraints over the features. The conjunction of the constraints expressed in an FM defines the set of all valid product configurations of an SPL [CW07].

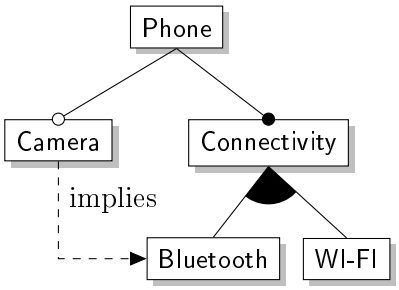


Figure 2.2: Example FM of a phone's features

Table 2.1: Configurations for the sample FM

Camera	Connectivity	Bluetooth	WI-FI
X	X	X	
X	X	X	X
	X	X	
	X		X
	X	X	X

FMs organize and structure concepts or features on multiple levels using a tree structure. The higher a feature is in the tree, and thus the closer it gets to the root of the tree, the more abstract it is. Conversely, the lower it is in the tree, the more concrete it is.
FMs also represent the variability between the features using mandatory or optional fea-

tures, OR-, XOR- and AND-groups. We will explain these concepts and their graphical notation later on. The hierarchy, the groups and the constraints can be translated into propositional logic and the translated propositional formula defines a set of legal configurations.
Table 2.1 lists these valid configurations for the example FM depicted in Figure 2.2 which represents a subset of the choice of features given to the users of a phone manufacturer.

An FM is actually a Feature Diagram (FD), which is formally defined in Definition 2.1, accompanied by a set of Cross-Tree-Constraints (CTCs). There exist two kinds of CTCs:

1. Implies constraints, which are binary implications between two features f_1 and f_2 ($f_1 \Rightarrow f_2$). An example of such CTC is shown in Figure 2.2;
2. Excludes constraints ($f_1 \Rightarrow \neg f_2$), which represent the fact that if feature f_1 is present, feature f_2 cannot be present.

DEFINITION 2.1
An FD is a tuple $FD\ (F,\ E,\ E_m,\ G_o,\ G_x)$ where

1. F is a finite set of features and $E \subseteq F \times F$ represents the set of edges from child features to their parents so that (F, E) forms a tree ;
2. $E_m \subseteq E$ is a set of mandatory edges ;
3. Go and Gx represent the sets of OR-groups and XOR-groups respectively. These are sets of non-overlapping subsets of E so that all the edges from the same subset of E share a common parent feature.

2.3.1 Notations

1. **Mandatory** The presence of the parent feature in a configuration implies the presence of the child feature. 
2. **Optional** The child feature may or may not be present in a configuration which contains its parent feature. 
3. **OR** If the parent is selected, then at least one of its child features must be selected. 
4. **XOR** If the parent is selected, exactly one of the child features should be selected. 

2.4 Data Mining

According to Han et al. [Han05], *"data mining refers to* extracting *or* mining *knowledge from large amounts of data"*. Most data mining techniques can be used to discover patterns within the data to analyze customer transactions, habits, ... or even discover fraud in financial data. These techniques include cluster analysis, anomaly detection and association rules mining. We are interested in those techniques because it enables us to identify features, and more importantly relationships between features in textual documents with little to no structure. We will focus on cluster analysis (Section 2.4.1) and association rules mining (Section 2.4.2) in this section as those are the two techniques that we use in our procedure.

2.4.1 Cluster analysis

Cluster analysis is a technique used in data mining to identify groups of elements that share a common aspect with each other but not with the rest of the data. This can be used to cluster customers in order to later target customer segments for marketing purposes. There are several different algorithms that can be used for clustering elements and we will give a brief introduction of the main ones.

2.4.1.1 Hierarchical Clustering

Hierarchical clustering is a data analysis method, the goal of which is to build a hierarchy between the clusters. There are two kinds of hierarchical clustering: (1) agglomerative clustering and (2) divisive clustering. The former is a bottom-up approach while the latter is a top-down approach.

Agglomerative clustering starts by creating a cluster for every element (also known as *data point*). The next step consists in merging the two closest clusters, and this step is repeated until all the small clusters have been merged into a single cluster. In order to determine which the two closest clusters are, the algorithm can use a variety of similarity metrics to evaluate the distance between two data points. Some commonly used metrics include the *Euclidean distance*

$$||a - b||_2 = \sqrt{\sum_{i=1}^{n} (a_i - b_i)^2} \tag{2.1}$$

the *Manhattan distance*

$$||a - b||_1 = \sum_{i=1}^{n} |a_i - b_i| \tag{2.2}$$

or the *cosine similarity*

$$\frac{a \times b}{||a||\,||b||} \tag{2.3}$$

where a and b represent two distinct data points. Those similarity metrics are then used in the linkage criteria to decide the clusters that should be merged. Again, there are several different criteria, and some of the most popular include *Single-linkage*

$$min\{d(a,b) : a \in A, b \in B\} \tag{2.4}$$

Complete-linkage

$$max\{d(a,b) : a \in A, b \in B\} \tag{2.5}$$

or *Group average*:

$$\frac{1}{||A||\,||B||}\sum_{a\in A}\sum_{b\in B} d(a,b) \tag{2.6}$$

where A and B are two different clusters and $d(a,b)$ is a similarity metric between two data points a and b.

Divisive clustering on the other hand starts by putting all the data points into one cluster and step by step divides this cluster into smaller ones. It uses the same metrics as the agglomerative clustering to choose the order in which the clusters are split.

2.4.1.2 k-means Clustering

The k-means clustering algorithm was introduced by MacQUEEN in 1976 [J.76]. In his paper he describes how to partition some data into k distinct sets based on their similarity. The first step of the algorithm consists in randomly selecting k centroids where k is the desired number of clusters. The next step is iterative, and at every iteration, all the data points are assigned to their nearest centroid. The metric used to determine if two data points are close is the Euclidean similarity:

$$d(a,b) = \|a-b\|^2 \tag{2.7}$$

The algorithm ends when the centroids stabilize, that is until it converges.
This algorithm is based on a heuristic approach, and it does not guarantee that the final k clusters are the optimal configuration.

2.4.1.3 Spherical k-means (SPKMeans) Clustering

The SPKMeans clustering algorithm [DM01] was introduced by Dhillon et al. in 2001. Their approach addresses both the quality of the solution and computational efficiency in the task of partitioning textual documents. It uses *cosine similarities* to compare documents represented in accordance with a *vector space model*.

The *cosine similarity* between two documents d_1 and d_2 is based on the angle between the term weight vectors representing d_1 and d_2 as follows:

$$d(d_1,d_2) = 1 - cos(d_1,d_2) = 1 - \frac{\langle d_1, d_2\rangle}{\|d_1\|\,\|d_2\|} \tag{2.8}$$

SPKMeans is a centroid-based clustering technique, just like the k-means clustering that we have introduced in the previous section. This means that each of the clusters is represented by its central vector called *centroid* (which does not have to be an instance in the cluster).
The SPKMeans problem consists in finding a documents partitioning into k clusters with similarity d that minimizes

$$\sum_{i=1}^{n}(1 - cos(d_i, p_{c(i)})) \tag{2.9}$$

over all assignments c of documents i to clusters $c(i) \in \{1, \cdots, k\}$ and over all centroids $p_1, \cdots, p_k$.

The k-means clustering approach seeks a similar minimization but uses the Euclidean similarity. Compared with the *cosine similarity*, the Euclidean similarity lacks a normalization regarding the documents length and tends to over-represent long documents.

The SPKMeans algorithm consists in two stages.

1. The first stage is similar to classical k-means clustering. The algorithm starts by randomly selecting k centroids where k is the desired number of clusters. Then, at each iteration, each of the instances is assigned to the cluster with the most similar centroid. At the end of the iteration, each centroid is recomputed based on the new set of descriptors constituting its cluster. The algorithm iterates until the centroids stabilize.

2. The second stage is an iterative process that incrementally optimizes the objective function [DM01]. At each iteration, one of the *feature descriptor* is randomly selected and moved to another cluster to increase the gain of the objective function. The centroids are then updated and this process continues until convergence.

2.4.2 Association Rules Mining

The concept of association rules mining was first introduced by Agrawal et al. in [AIS93] in the context of customer basket analysis. They designed an algorithm to find association rules between items in a large database of customer transactions (e.g., $bread \Rightarrow milk$ which indicates that a customer who bought bread will most likely be buying milk as well).

The input of the association rule mining algorithm is a dataset containing transactions. A transaction t is a set of items $I = i_1, i_2, \cdots, i_n$. In the context of basket analysis, those transactions can be thought of as the records of the basket contents that each client has bought.

Association rules mining is a two-step process. The first step is the frequent itemsets mining, and the second step is the association rules generation based on those itemsets. There are several algorithms both for finding itemsets and generating association rules.

In Section 2.4.2.1 we will discuss the algorithms for finding frequent itemsets while in Section 2.4.2.2 we discuss the algorithms for generating the association rules.

2.4.2.1 Frequent Itemsets Mining

Frequent itemsets mining is a problem that has been extensively studied, with algorithms such as Apriori [AS94], *FPGrowth* [HPY00] or Eclat [Zak00]. We will focus this discussion on the Apriori and the *FPGrowth* algorithms as these are the two most commonly used.

Apriori This algorithm, introduced by Agrawal et al. [AS94], discovers large itemsets using a multi-pass approach over the data. During the first pass, it counts the support of all the items within the data and keeps only the ones that are large (i.e., whose support is higher than the minimum support). Those large itemsets are called *seeds* and are used in the subsequent passes to generate the new potentially large itemsets that will later be used, in their turn, as *seeds* if their actual support is higher than the minimum support. They call those potentially large itemsets *candidate itemsets*.

FPGrowth This algorithm, introduced by Han et al. [HPY00], first constructs a Frequent-pattern tree (FP-Tree) (1) and then generates the frequent itemsets based on that FP-Tree (2).

(1) The FP-Tree is constructed from the input data by processing every *transaction*, finding the support of every item only to collect the frequent ones and then sorting those frequent items. Then, a root is added to the FP-Tree with the label "null". Finally, for every *transaction*, its frequent items will be sorted according to the ordered list that was previously constructed and then added to the tree in the following way:
The first item of the *transaction* is added to the tree as a children of the root node. If there was already a node in the tree with the same name as this item, its frequency count will be incremented; if not, a new node will be created and in both cases the remaining items of the *transaction* will be added under the node as a branch. Every time a new node is added to the tree, it will be linked to its successor nodes (in pre-order) that share the same name.

(2) To generate the list of frequent itemsets from the FP-Tree, the algorithm starts from the leaves and makes its way towards the root. It will extract all the prefix path sub-trees for every item within the FP-Tree. Those prefix path sub-trees will then be processed in order to extract the frequent itemsets.

In their paper, Han et al. [HPY00] ran experiments to compare their algorithm (*FP-Growth*) and the Apriori algorithm. Their results show that *FPGrowth* runs significantly faster than Apriori when *"the dataset contains an abundant number of mixtures of short*

and long frequent patterns" which, as we will explain later in this document, is representative of our dataset. Another advantage of this algorithm over the Apriori is that, due to the FP-Tree structure, it is very efficient memory-wise and scales up well.

2.4.2.2 Association Rules Creation

Agrawal et al. explain in [AIS93] their procedure to mine association rules. As we mentioned earlier, this is a two-step process and the first step was explained in the previous section. What needs to be done now is to generate all the association rules that use the items of the frequent itemsets. Let us say we have an itemset $F = I_1, I_2, \ldots, I_k$, then we need to generate at most k rules containing those items. Association rules are found in the following form:

$$A \Rightarrow B \tag{2.10}$$

where A contains at most $k-1$ items, B contains the remaining items $(F \setminus A)$ and $A \cap B = \emptyset$. Every association rule has a *support* which indicates the proportion of transactions in which both A and B appear:

$$support(A \Rightarrow B) = \frac{S(A \cup B)}{S} \tag{2.11}$$

where S is the multiset of all transactions and $S(A \cup B)$ is the multiset of transactions that contain both A and B. Association rules with various supports will be found but only the ones above a certain threshold are of interest to us.

Another measure used in the association rules is the *confidence*. It indicates the proportion of transactions that contain itemset B among the transactions that contain the itemset A.

$$confidence(A \Rightarrow B) = \frac{support(A \cup B)}{support(A)} \tag{2.12}$$

Chapter 3

Feature Model Mining

In this chapter we will perform an analysis of state-of-the-art FM mining techniques. We first look at work related to FM mining from logical formulas and formally defined datasets. Then, we discuss existing work related to FM mining from informal and textual product descriptions.

3.1 Feature Model Mining from Formally Defined Datasets

In their paper, Czarnecki et al. [CW07] focus on extracting FMs from logical formulas. They give a semantics to FMs using boolean logic and discuss the challenges and requirements of FM extraction. In their discussion, they point out that many different FMs can be extracted from the same logical formula. They show how to represent a propositional formula that is in conjunctive normal form (CNF) with an IG, which is a representation of the dependencies between features in product configurations, and how various FMs, which are logically equivalent but graphically different, can be extracted from the same IG. From this observation, the authors provide some considerations about FM structuring and FM visualization. If many FMs, each with a different representation, can be extracted from the same logical formula, then the logical structure is not the only structuring criterion for FMs. Additional information, such as manual structuring, could be used to sort out the adequate structure. The authors also underline that an automatically extracted FM should expose a maximum of logical structure and avoid redundancies.

They propose an FM extraction algorithm from logical formulas. The algorithm computes an IG as well as AND-groups, OR-groups and XOR-groups. The implementation relies on *Binary Decision Diagrams* to perform operations on logical formulas. BDDs are representation for logical formulas that can support operations such as satisfiability or tautology checking efficiently.

The translation of FMs in boolean logic and the FM extraction algorithm from logical formulas offer a support for refactoring operations on FMs. However, the authors point out that they have only considered logical information to structure FMs and that considering additional structuring information should be investigated in future work.

In [CSW08], Czarnecki et al. introduce PFMs as an extension of FMs and discuss how to mine such models from a given set of product configurations. Probabilistic logic is used to formalize PFMs as sets of logical formulas. The authors distinguish two types of constraints forming the set of probabilistic logical formulas :

1. *Soft constraints* which express probabilistic dependencies between features. Soft constraints are of the form "f_1 encourages f_2" which means that there is a high probability that feature f_2 is selected when feature f_1 is selected. For example, a soft constraint can express that 80% of the product configurations having feature f_1 also have feature f_2 (written f_2 *given* f_1 *[80%]*).
2. *Hard constraints* which are constraints with a confidence of 100%.

A PFM consists of an FD, which encompasses hard constraints, and an additional set of hard and soft constraints. PFMs determine a probability distribution on product configurations.

The article proposes an algorithm to mine PFMs from sets of product configurations. The algorithm uses data mining techniques to discover binary feature implications, OR-groups and XOR-groups implications and binary feature exclusion clauses. The techniques used are conjunctive association rules mining and disjunctive association rules mining. A minimal confidence threshold can be defined for this mining phase so that all the rules that have a confidence above the minimal threshold are found. The algorithm builds an FD based on the mined binary feature implications, group implications, and exclusion clauses with confidence of 100%. The mined rules with confidence lower than 100% are then gathered into a set of additional soft constraints aside the FD.

A PFM can guide the users in their selections of features. These constraints can be used for a configuration tool as they express preferences for features to select and can change dynamically. For example, if feature f_2 is present in 20% of the product configurations, it will not be initially recommended. However if the user selects feature f_1 for his configuration and if there exists a soft constraint f_2 *given* f_1 *[80%]* then the tool may recommend feature f_2.

In [ACSW12], Andersen et al. discuss the mining of FMs from propositional constraints. First the authors formally define the FM synthesis problem and discuss its complexity which is NP-hard. The input of the FM synthesis process is a set of feature dependencies or a set of product configurations. The output is a Feature Graph (FG) which, unlike an FD, does not necessarily have a tree-structure. The FG represents dependencies between features along with propositional constraints expressing exclude-edges, OR-groups, XOR-groups and MUTEX-groups. The authors separate the FM synthesis process into two reusable steps :

1. The elicitation of the Directed Acyclic Graph (DAG) hierarchy for the FG.
2. The identification of groups and the recovery of CTCs that cannot be expressed in the feature hierarchy.

The authors give different examples of abstract workflows for FM synthesis procedures that involve the mining of an FG and the extraction of an FM from the FG. An FG is a representation of all possible FMs that could be sound results of the synthesis. Some of the features in the FG may have multiple parents. Thus, selecting an FM from the FG consists of choosing a tree-structure from the FG. The authors state that, in general, the tree selection requires an *additional input* such as user decisions or a feature similarity measure. However they do not offer a reusable procedure to solve this step of the procedure.

The article presents an algorithm for synthesis of FMs from formulas in conjunctive normal form (CNF) and from formulas in disjunctive normal form (DNF). The efficiency of these algorithm is then evaluated. The quality of the extracted FMs is not addressed as the authors are only concerned about the synthesis of the FG which is the representation of all potential FMs. The evaluation of the quality of the derived FMs would depend on the techniques used to select FMs from FGs. Examples from public repositories and randomly generated models are used for the evaluation which show that both the proposed algorithms outperform the old Binary-Decision-Diagram implementation by a factor of 10 to 1000 times. Regarding the scalability, the CNF algorithm scales to up to above 5000 features when the OR-group computation is switched off (it is the most difficult step) while the BDD-technique usually does not scale up to 2000 features.

In [SLB+11], She et al. propose a reverse-engineering procedure to extract FMs from sets of dependencies among features for operating systems (Linux, FreeBSD, eCos). During the construction of the hierarchy for the FD, they use an interactive approach to recommend the user with the likely parent candidates for a given feature. Their *parent ranking heuristics* is based on dependencies between features and similarities between feature descriptions.

In [ACP+12], Acher et al. present a semi-automated procedure to extract FMs from product descriptions expressed in a tabular format. The input for the extraction procedure is a table in which each row represents a product from the product line and each column has a label that will be used as a feature name in the extracted FM. The cells contain values that generally correspond to features of a product but theses values also capture information about the variability between the different features. For example, a value may be a list of potential features such as *Files, Database* or "*Yes*" which could indicate that the feature represented by the label of the column is mandatory. The format for the values in the input table does not have to be formally defined but the extraction procedure still requires the set of product descriptions to be described in this tabular format. The authors identify five variability patterns that can be detected from the cells values. The article also introduces *VariCell*, a dedicated language aimed to provide a practical solution to parse, scope, transform and structure a set of product descriptions

into an FM. The user can rely on *VariCell* to specify directives for data transformation and variability interpretation.

The FM synthesis process first extracts an FM from each product description and then merges them into a new FM that represents the sets of product descriptions. The merged FM should represent the union of the sets of configurations represented by all the FMs and preserve as much as possible their hierarchies. Firstly, the algorithm computes the hierarchy for the merged FM. Secondly, it computes the propositional formula that the FM should represent from the set of configuration. Finally, it uses propositional logic reasoning techniques to construct an FD based on the previously found hierarchy and propositional formula.

3.2 Feature Model Mining from Textual Product Descriptions

In [WCR09], Weston et al. discuss the difficulty of identifying features in textual documentation and address the problem of building an FM once the features are known. They highlight the need for tool support for these tasks and propose an FM construction framework. They consider features as clusters of related requirements and use statistical methods to determine similarity between the texts of the requirements. The compared requirements are these as demarcated in the input textual document. However, if the text is unstructured, the user has either to manually structure the document or specify the amount of sentences the document is going to be divided into in order to approximate the requirements. Requirements measured as most similar to each others are clustered together as features following a stepwise variant of Hierarchical Agglomerative Clustering ([HC06]). At each step, similar features and requirements are clustered together. The aggregation of small features leads to the creation of parent features. The features end up structured into a tree based on their similarity and the user can specify the maximum number of levels in the tree. The tool offers to display the feature tree which shows the associations between requirements and features. The user can add, remove and manually name the features based on his understanding of the domain. In the next step, the authors use the EA-Miner tool [SCRR05] which was originally developed for identification of cross-cutting concerns in textual documents. The tool parses the text to discover variability points, based on a variability lexicon and a grammatical pattern identifier. Once a variability element has been considered relevant, the analyst needs to decide how it has to be included in the FM.

In [HC06], Chen et al. present a semi-automated approach to building FMs based on requirements clustering. Again, features are considered as clusters of related requirements. Firstly, an undirected graph that models relationships between requirements is built from a list of requirements. The graph is called requirements relationship graph (RRG). The authors define five types of relationships between requirements; relationships are established between requirements, in case they share a common resource, or

when they require each other services, or if they specify similar behaviours. Secondly, a clustering algorithm is applied in the RRG to identify and organize features. These steps are executed for several sample applications so that one application feature tree is built for each one of them. Finally, the application feature trees are merged as a single domain feature tree.

Part II

Algorithm description

Chapter 4

Overview

In this chapter we address the limits of the techniques that can be related to our work in Section 4.1 and then use these limits to motivate our contribution in Section 4.2. Section 4.3 presents an overview of the FM mining procedure described in the following chapters.

4.1 Limitations of Related Work

Most existing work address the FM synthesis problem for a known logical formula or a known set of configurations. These algorithms require the products to be already described as subsets of a list of features common to all the products. [ACP+12] offers a semi-automated approach to extract FMs from product descriptions. The descriptions do not have to be formally defined but are still required to be structured into a tabular representation.

Moreover, the existing algorithms do not explicitly address the extraction of FMs from incomplete or inaccurate datasets. [WCR09] proposes an FM extraction procedure for textual descriptions of products but it requires some human intervention to incorporate variability information in the FM and it totally ignores incomplete descriptions. [WCR09] and [HC06] assume a certain structure in the product description or other knowledge that is exploited to hierarchically organize the features. They base their approach on detecting grammatical patterns to discover variability information to represent in the FM and do not rely on statistical analysis of features occurrences in product configurations. A statistical approach can discover information related to variabilities and commonalities between the products transversely to a large collection of descriptions without assuming the presence of specific grammatical patterns or the presence of words from a lexicon. It also allows to take the incompleteness of the data into consideration during the structuring phase of the FM.

An important limitation of prior work is the identification of the feature hierarchy; It does not directly address the problem of automatically selecting an FM with a meaningful hierarchy and often defers the problem to human intervention. In [CW07] and [ACSW12], the authors calculate a diagrammatic representation of all possible FMs. However they do not offer solutions to the problem of selecting a unique FM with a meaningful hierarchy. In [SLB+11], She et al. propose heuristics for identifying the likely parent candidates for a given feature in order to assist users in selecting a hierarchy. However the heuristics are specific to targeted operating systems (Linux, FreeBSD, eCos). Furthermore She et al. reported that their attempts to use clustering techniques did not produce a single and desirable hierarchy. They gave a possible reason, arguing that "*there is simply not enough information in the input descriptions and dependencies*" for the kinds of artifacts they considered.

While the existing work on FM mining mostly focuses on accurately representing known dependencies between features, it offers very few insight about ensuring the maintainability of the extracted FMs.

In [CSW08], Czarnecki et al. introduce PFMs made of soft and hard constraints. Soft constraints are formally described and indicate the conditional probabilities between the presence of features in configurations which enable reasoning about preferences between feature selections. Hard constraints express configuration rules that must be obeyed by all the products, soft constraints should be respected by most configurations but can be violated by some of them. The authors propose an extraction procedure that relies on association rule mining. The prior work requires the product configurations to be formally defined as sets of features and it assumes the completeness of the data. The authors do not address the use of probabilistic approaches to derive feature hierarchies and variability information from an incomplete dataset.

4.2 Contribution

This thesis presents a fully automated process to mine FMs *from informal textual product descriptions*. It also analyses the application of the process to product descriptions that are publicly available on software repository websites

The mining of the descriptions and the discovery of variability points do not depend on any phrase structure patterns or on a predefined lexicon. Our approach for product descriptions mining relies on a vectorial representation of documents and statistical comparisons of the vectors. The structuring of the FMs and the discovery of variability points rely on data-mining and text-mining techniques.

The FM extraction procedure has been applied on data from software descriptions repositories publicly available online. These repositories form the data corpus for the evaluation of the process.

The procedure contains a feature mining phase which automatically computes names for features from their textual descriptions collected from the repositories. The algorithm does not require a specific structure for the descriptions and does not require a common structure between descriptions for different products.

The process addresses the problem of mining FMs from *incomplete* datasets. An incomplete dataset is made of product configurations for which features that are not notified as present may actually be present (i.e., the description of the products are not exhaustive). While most related work on FM mining assume working with a complete *product-by-feature matrix* describing accurately the product configurations, we use data-mining techniques in combination with text-mining techniques to address the incompleteness and the uncertainty of the data in order to provide an approximation of a FM representative of the considered product line.

The process is *fully automated*. Most related work suggest a manual intervention, especially to decide the hierarchical and tree structure of the FM. In [ACSW12], Andersen et al. compare the decision of the structure of the FM to a spanning tree type of problem that requires an *additional input*. Instead of a human intervention as additional input, we use statistical analysis and text-mining techniques to compute the structure of the FM so that the process is able to suggest an FM on its own. We present the application of two text-mining techniques to structure a mined FM; clustering the features so that features related to a common aspect of the domain are grouped together in a branch of the resulting FM (1) and an automated generation of new *abstract features* in order to enhance the readability of the FM (2). The application of these two techniques has been inspired by the way humans build FD hierarchy from a list of features.

4.3 Algorithm Overview

The steps of the algorithm we designed are depicted in Figure 4.1. The algorithm consists of three phases (shown in Figure 4.2) that we briefly present here as an overview before going into more details in the remaining chapters. The first phase is the mining of features, the features are extracted from informal product descriptions. We delve into a little more details in Section 4.3.1. The second phase is the mining of an IG, for which we provide a brief overview in Section 4.3.2. The third phase is the structuring of the FM and is described in Section 4.3.3

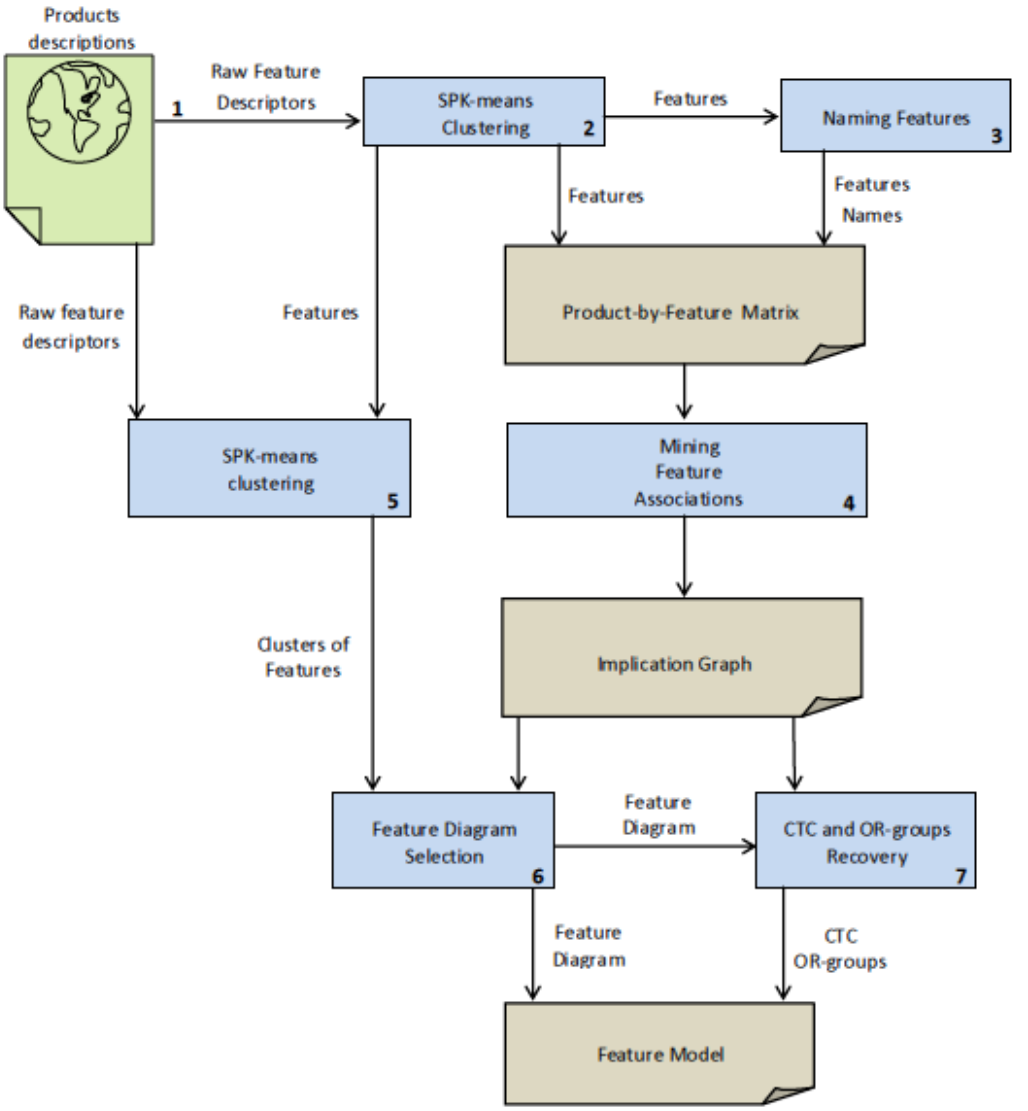


Figure 4.1: Fully automated process that mines FMs from textual product descriptions

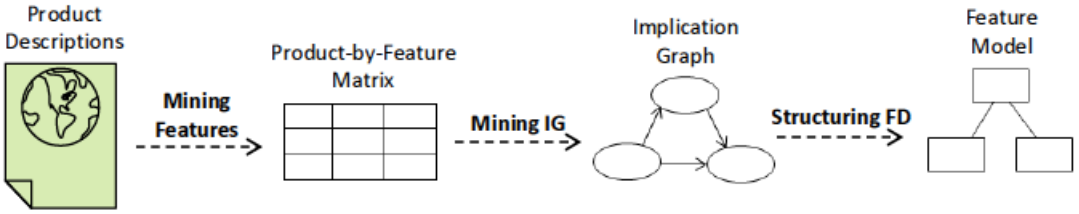


Figure 4.2: The FM mining algorithm consists of three phases.

4.3.1 Mining Features

The input of the algorithm are the textual descriptions of products. The first phase consists in representing these descriptions as a *product-by-feature matrix*.

Step ❶ is the initial step of the process, during which we mine informal product descriptions on online software repositories using the Screen scraper utility[1]. This step is formally described in Section 5.1.

In step ❷, the mined *feature descriptors* are processed in order to identify features. We will explain how we use clustering on the *feature descriptors* to find the features in Sections 5.2 and 5.3.

In the last step of this first phase ❸ meaningful names are automatically found for every features and then a *product-by-feature* matrix is built to represent the products as a set of configurations of features. This step is explained thoroughly in Section 5.4 and 5.5.

The feature mining phase is presented in details in Chapter 5.

4.3.2 Mining the Implication Graph

The second phase of the process consists in mining all the logical constraints that will be encompassed in the FM from the product-by-feature matrix that has been constructed in the previous phase. We use data mining techniques in step ❹ on the matrix to find association rules between the features (lines 2-5 in Algorithm 1). This step is explained in Section 6.1. These association rules are used to create an IG. The IG is a directed graph in which the edges between two features f_1 and f_2 represent an association rule between the two features. The creation of the IG is explained in greater details in Section 6.2.

Other logical constraints among features are mined in order to enhance the FM with AND-groups and OR-groups later on. The discovery of theses constraints is explained in Section 6.3 and Section 6.4

[1] http://www.screen-scraper.com/

Algorithm 1 Feature Model extraction procedure

1: ▶ Association rules mining
2: **function** AssocRules(P, MIS)
3: $F \leftarrow CFP - Growth(P, MIS)$ ▷ Frequent Itemsets
4: $A \leftarrow Agrawal(F, MinSup)$
5: **return** A

6: ▶ Implication graph
7: $G(V, E) \leftarrow IG(AssocRules(Configurations, MIS))$

8: ▶ Clustering the features
9: $C \leftarrow SPKMeans(Features)$

10: ▶ Building FDs for the clusters
11: **for** $i \leftarrow 1$ to n **do** ▷ Number of clusters
12: $G_i \leftarrow SubGraph(G, C_i)$ ▷ Cluster i
13: $SCC_i \leftarrow StronglyConnectedComponents(G_i)$
14: $FD_i \leftarrow FeatureDiagram(G_i, SCC_i)$

15: ▶ Aggregating the FDs
16: $A \leftarrow AssocRules(ConfigsClusters, MISClusters)$
17: $G \leftarrow merge(\{FD_1, \ldots, FD_n\}, A)$
18: $SCC \leftarrow StronglyConnectedComponents(G)$
19: $FD \leftarrow FeatureDiagram(G, SCC)$
20: $FD \leftarrow PrimeImplicates(FD)$

21: ▶ Recovery of CTCs and OR-groups
22: $CTC \leftarrow G - MG$ ▷ Cross-Tree Constraints

4.3.3 Structuring the Feature Model

In step ❺, the features are partitioned into clusters as explained in Section 7.2 (line 9 in Algorithm 1). In step ❻, a hierarchy between the features which form a FD is extracted from the IG by using data-mining techniques. The computation of the tree structure of the FD consists in two steps. Firstly, a hierarchical structure is computed between the features belonging to the same cluster (lines 11-14 in Algorithm 1). Secondly, a structure is found between the different clusters (lines 16-19 in Algorithm 1). These two steps are respectively described in Section 7.3 and Section 7.4.

In the final step, step ❼, we identify the CTCs that have been left out while structuring the FD (line 22 in Algorithm 1) and also identify the OR-groups that can be graphically represented in the FD (line 20 in Algorithm 1). This step uses both the IG generated in step ❺ and the FD. The FD, CTCs and OR-groups form the final FM.

Structuring the FM is a very important part of the process and it is explained in Chapter 7.

Chapter 5

Mining Features from Product Descriptions

The first phase in the extraction of an FM from informal product descriptions is the elicitation of the features themselves. In this chapter we explain the process created by Dumitru et al. in [DGH+11] that we use for extracting features from a software download platform such as SoftPedia[1] and CNET[2]. Those websites list thousands of different software products that can be downloaded for free. We will be using SoftPedia as an example throughout this part to describe and then later validate our approach but any other website or source of information listing product descriptions would work.
SoftPedia is structured in categories (Antivirus, Compression tools, iPod tools, ...), and every software product belongs to a category and has its own page listing various information about it (e.g., user rating, platform, key features, ...).

On SoftPedia, two different descriptions for two distinct products can use different words and sentences to cover what is actually the same feature. In other words, the authors of the product descriptions are not constrained by a predefined list of feature names to use but are free to describe features the way they want. The aim of the feature mining phase is to identify different feature descriptors among all the products that actually cover the same feature and then to automatically assign them a common feature name. At the end of this phase, a list of feature names has been identified and all the products can be described as a set of features selected from the list.

This chapter starts by explaining how the process collects the *raw feature descriptors* in Section 5.1 and how they are preprocessed in Section 5.2. Section 5.3 is about clustering similar *raw feature descriptors* together while Section 5.4 describes how the feature names are selected for each of the clusters. Figure 5.1 shows an overview of the features mining

[1] http://www.softpedia.com
[2] http://download.cnet.com/windows/

phase.

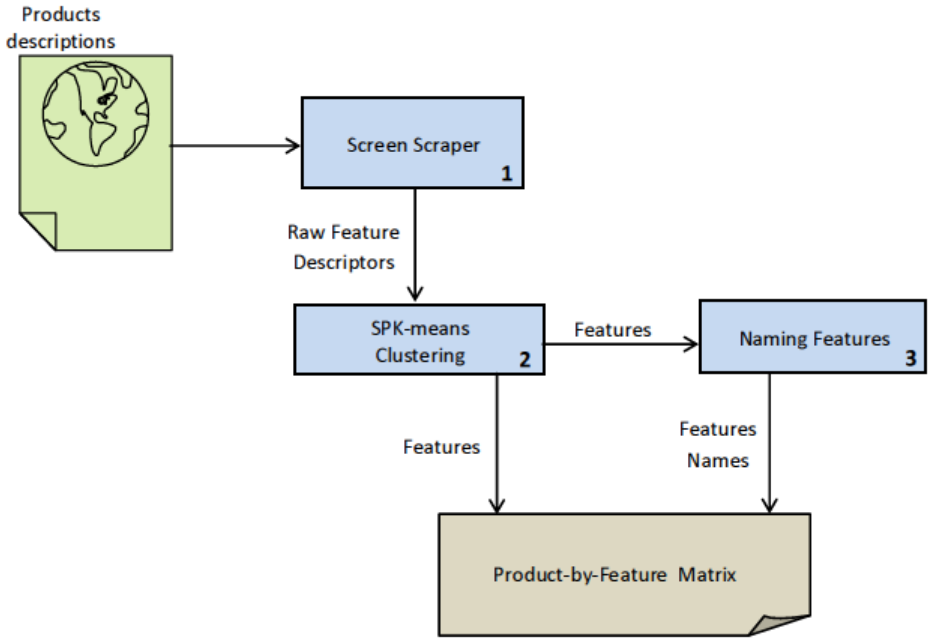


Figure 5.1: Feature mining phase overview.

5.1 Collecting *raw feature descriptors*

In the first step of this process, we collect the *raw feature descriptors* for a given category (in this case, antivirus) from http://www.softpedia.com. We use the Screen scraper utility, which is a tool that lets a user define HTML tags to filter out the information inside a web page, to visit the page of every product in the category and scrape them to extract the information that can be useful to identify feature names. The pages contain a general description of the product and a bulleted list of its key features (see Figure 5.2).

Figure 5.2: Screenshot of Microsoft Security Essentials' ©product page as of February 2013 showing the description and key features of an antivirus software

Table 5.1: Example of key features from a SoftPedia Antivirus Product

Safepay	Keeps hackers at bay by automatically opening all your online banking pages in a separate, secure browser.
Dashboard	See all the status and licensing information about your software and services in your own, MyBitdefender dashboard. Now accessible from anywhere, anytime, from any Internet-connected device.
Security Widget	Enables you to keep track of all of your security related tasks, plus lets you quickly and easily drag-and-drop files for quick scanning for viruses right from your desktop!
Parental Control	Blocks inappropriate content, restricts Web access between certain hours, and helps you remotely monitor your children's online activity even on Facebook!
USB Immunizer	Immunizes any Flash Drive from viruses, when they are connected to your computer, so that you never worry again about USBs infecting you or your friends.
Active Virus Control	A proactive, dynamic detection technology which monitors processes behavior in real-time, as they are running, and tags suspicious activities.

Raw feature descriptors are collected from these pages by parsing the software descriptions into sentences and adding to it the items in the bulleted list of features. The problem with the description and the key features is that they have been written in natural language without respecting any predefined structure. In other words, for two different products, an identical feature may have been described in two very different ways. For instance, a feature regarding the updating of an antivirus is listed in various products as:

1. *"Automatic updates";*
2. *"Virus definitions are automatically updated when your computer is connected to the Internet";*
3. *"Continuous updates: round-the-clock updating of the definitions database";*
4. *"Updates: Fast application of updates";*
5. ...

5.2 Preprocessing

Once these *raw feature descriptors* have been extracted, they need to be preprocessed before we can start working on them. The purpose of this preprocessing phase is to narrow the *raw feature descriptors* to relevant terms and to give them an easy-to-process representation. This preprocessing phase can be divided in smaller steps as follows:

1. Stemming the words to their morphological root;
2. Removing stop words (using a list of very common words) such as "the", "is", "on';
3. Building a vector for each *feature descriptor* in which the index corresponds to a word of that descriptor and the values correspond to the Term frequency-inverse document frequency (tf-idf) score of the word. This representation of textual artefacts as vectors of term weights is called *vector space model* based representation.

The first step (stemming) is achieved through the use of Porter's stemming algorithm [Por80]. This algorithm removes the suffixes of the words to obtain their morphological root so that they can be more easily compared.
The next step (removing stop words) is simply filtering out common words from the descriptors using a publicly available list[3].
The final step (building the vector) consists in attributing a weight to every remaining term in the descriptor. These weights are computed using tf-idf as shown in the following formula:

$$w_{t,d} = tf_{t,d} * log(\frac{N}{df_t}) \tag{5.1}$$

[3] e.g., http://www.ranks.nl/resources/stopwords.html

where, $tf_{t,d}$ is the number of occurrences of term t in descriptor d, df_t is the number of descriptors that contain term t, and N is the total number of descriptors.

The formula shows that the more a term t occurs in a descriptor d and the smaller the set of descriptors featuring t is, the higher the weight of t in vector of descriptor d. In other words, a term t is representative of a descriptor d if t is frequent in d and if t is specific to d.

5.3 Clustering the Feature Descriptors

Once the *feature descriptors* have been found and preprocessed, they need to be partitioned into clusters. The idea is to group together *feature descriptors* that actually stand for the same feature. The algorithm used to do this is a two-stage SPKMeans clustering algorithm [DM01] which has previously been shown to perform well for clustering features [DGH+11].

An example of *feature descriptors* which have been clustered is available in Section A.1 on page 97.

5.4 Feature Name Selection

After having clustered the *feature descriptors* together in the previous step, we need to find a name that is representative of the topic captured in the different features and also meaningful as they will be presented to the users in the final FM.

The cluster-naming process was developed based on informal experimentation. This process comprises the following steps: (1) selecting the most frequently occurring phrase from among all of the *feature descriptors* in the cluster, (2) discovering *frequent itemsets* and finally (3) selecting the best name. Hu et al. introduced a similar method for "summarizing customer reviews" [HL04].

Most frequently occurring phrase In order to select the most frequently occurring phrase in the *feature descriptor* we use the Stanford Part-of-Speech (POS) tagger[4] to tag the different terms in the *feature descriptors* with their POS. However, not all the terms within a *feature descriptor* are useful. Therefore, we retain only nouns, adjectives and verbs.

[4] http://nlp.stanford.edu/software/tagger.shtml

Discovering *frequent itemsets* We need to find the sets of terms that most frequently occur together in the *feature descriptors* for every cluster we found previously. Those sets of terms are called *frequent itemsets*. There are many algorithms that can be used for mining those *frequent itemsets* (Apriori [AS94], Frequent Pattern Growth (FPGrowth) [HPY00]). The one we chose to use in this case is *FPGrowth* because of its efficiency. Indeed, due to the large quantity of data we need to process, a memory-efficient algorithm was a logical choice.

Selecting the best name For every cluster we select the frequent itemsets, found in the previous step, of maximum size and add them to the set FIS_{max}. Then, for every *feature descriptor* in the cluster, we select the shortest sequence of terms as a possible name. For example, let $FIS_{max} = \{prevents, intrusion, hacker\}$. For a given *feature descriptor* such as: *prevents possible intrusions or attacks by hackers trying to enter your computer*, the selected candidate name is *prevents possible intrusions or attacks by hackers*. Out of all the candidate names, we pick the shortest one as it will be used in the FMs and therefore must not be too lengthy.

5.5 Product-by-feature matrix

So far, products have been described as lists of *raw feature descriptors*. Two different *raw feature descriptors* for two different products can actually represent the same feature but now that the *raw feature descriptors* have been clustered to identify feature names, we can use these names to describe the products as sets of features.

When all the products are described as sets of features, they can be represented as a *product-by-feature matrix*. In this matrix, columns represent features and lines represent products. If the cell at line l and column c, is filled with a 1, it means that product from line l has feature from line c. Conversely, if the cell at line l' and column c', is filled with a 0, it means that product from line l' does not have feature from line c'. Table 5.2 shows a fraction of a product-by-feature matrix mined from SoftPedia.

Active detection of downloaded files	Active detection of instant messenging	Active detection of removable media	Active detection of search results	Anti-Root kit scan	Automatic scan	Automatic scan of all files on startup	Automatic updates	Behavioral Detection	Command line scan	Contain viruses in specific quarantine	Customized firewall settings	Data encryption	....	
														a-squared Free
	●	●			●								●	Acronis AV
	●	●											●	AhnLab Platinum
	●												●	Ahnlab
							●		●					Anti-Trojan Elite
				●										AppRanger
														Ashampoo a.m.
	●												●	Auslogics AV
	●	●	●	●	●	●	●	●	●		●		●	Avast! Pro AV
				●					●		●	●		Avast! Int. Sec
	●	●	●	●	●		●						●	AVG AV Pro.
	●	●	●	●	●		●						●	AVGAV+Firewall
					●	●	●						●	Avira AntiVir Premium
					●	●	●	●	●		●			Avira SmallBusiness Suite
					●	●	●	●	●		●			Bkav2008
	●	●	●	●		●		●	●		●		●	BitDefender Total Sec. '10
	●	●	●	●		●		●	●		●		●	BitDefender Int. Sec. '10
				●										Corbitek AntiMW
				●					●					CyberDefender Int. Sec.
					●									COMODO Cloud Scanner
				●										Dr.Web
	●	●	●	●	●		●	●	●		●		●	G DATA AV '10
			●		●		●							Fix-it Utilities Pro.
			●		●	●								Gucup AV
														GSA Delphi Induc Cleaner
						●	●							Hazard Shield
														GGreat Owl USBAV
				●		●	●	●						Jiangmin AV KV '10
							●							K7 AV
					●		●							Immunet Protect
														MW Destroyer
														Kaspersky Ultra-Portables
							●							Mx One AV
														MultiCore AV AntiSpyware
				●	●			●	●					McAfee VirusScan
														Microworld AV Toolkit Util.
	●	●		●	●	●		●	●		●		●	Norton Int. Sec.
				●										Novashield - Anti MW
							●	●						NoVirusThanks MW Rem.
							●							Norman Virus Control
														Norton AV 2009 GamingEd.
							●		●	●	●		●	Norton AV
										●	●		●	Norman Sec. Suite
							●	●					●	Network MW Cleaner
							●	●	●				●	PC Tools Int. Sec. '10
	●	●					●	●			●		●	Outpost Sec. Suite Pro
	●						●							nProtect GameGuard Pers.
											●		●	Quick Heal Int. Sec. '10
											●			Quick Heal Total Sec. '10
								●	●					Steganos Int. Sec. 2009
														Steganos AV 2009
														SpyDLLRem.
														Tizer Rootkit Razor
							●						●	The Shield Deluxe '10
							●			●	●			The Cleaner 2011
							●							SystemSuite Pro.
						●			●					SysIntegrity AM
	●			●	●	●	●	●	●				●	VIPRE AV Premium
						●	●							VirIT eXplorer Lite
	●					●	●							TrustPort PC Sec. '10
		●		●	●		●				●		●	Twister Anti-TrojanVirus
	●					●							●	ZoneAlarm Sec. Suite
							●							Wlording AntiSpyware
											●			Your Free AntiSpyware
						●			●					Webroot Int. Sec.
						●	●	●						Windows AV Mate Prof.
														Wimp
							●							VIRUSfighter
	●					●	●	●			●			VirusBuster Pro.
						●	●	●			●			VirusBuster Personal

Table 5.2: Part of a feature-by-product matrix mined from SoftPedia

Chapter 6

Implication Graph Mining

An FM is a graphical representation of a logical formula that comprises the different configuration constraints between features (see Section 2.3). These configuration constraints are, in general, of the form "*the presence of feature* f_1 *in a product configuration implies the presence of feature* f_2" ($f_1 \rightarrow f_2$) or "*the presence of feature* f_1 *in a product configuration implies the presence of either feature* f_2 *or feature* f_3" ($f_1 \rightarrow f_2 \vee f_3$). Thus, the logical formula represented by the FM defines the constraints in terms of features co-occurrences that a product configuration must satisfy to be valid regarding the FM. In this chapter, we are interested in the mining of such a logical formula from a dataset (i.e., a set of product configurations). From now on, we refer to the product-by-feature matrix obtained at the end of Chapter 5 as the *dataset*.

Firstly, the IG mining phase consists in applying data mining techniques on the dataset to discover the disjunction made of all the logical constraints that the future FM will have to encompass. Secondly, an IG is built to give the mined constraints a graphical representation.

In this chapter as well as in the following one, we use the list of features given in Table 6.1 to illustrate the various steps involved in the mining of a FM from a dataset of antivirus products. The names of these features were manually assigned for clarity purposes.

In Section 6.1 we describe the mining of the association rules which are used to create the IG in Section 6.2. Figure 6.1 shows an overview the features mining phase.This chapter also describes the techniques used to mine AND-groups and OR-groups for the final generated FM.

As this chapter treats the discovery of the logical formula that will be expressed by the final generated FM, we include the descriptions of the procedures to mine OR-groups and AND-groups. Section 6.4 describes the mining of minimal disjunctive clauses to create OR-groups and Section 6.3 deals with the detection of Strongly Connected Components

Table 6.1: Sample features for an antivirus product line

Sample features
anti-spyware
mail content scanning
files scanning
removes keylogger
mail attachement analysis
removes trojan
antispam
behavior based detection
anti-rootkit

(SCCs) to create AND-groups. However, due to the design of the algorithm, these techniques are only applied during the third phase that deals with the structuring of the FM and which is explained in details in Chapter 7.

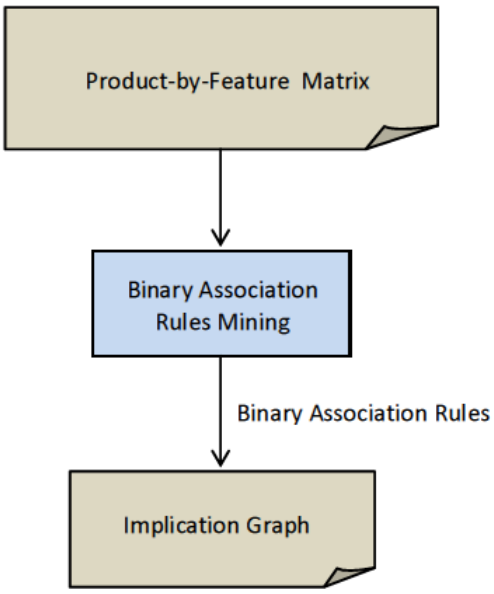


Figure 6.1: IG mining phase overview.

6.1 Association Rules Mining

To mine association rules, we use the product-by-feature matrix (see Section 5.5) as the dataset. Each line of the matrix represents a product and lists the feature that this product has and is thus considered as a transaction in the association rules mining process. However, as mentioned earlier, the data in this matrix is incomplete due to the fact that it has been mined from software product descriptions that assumedly only list their key features. Thus, a product can have many more features than the ones listed in its description and the absence of a feature in the matrix does not mean that the product does not have that feature, it only means that this feature was not listed for that product.

Mining the association rules is a two-phase process which starts by finding the frequent itemsets (see Section 6.1.1) and then generates the association rules based on the frequent itemsets that were previously found (see Section 6.1.2). Section 6.1.3 addresses the problem of our incomplete dataset.

6.1.1 Finding Itemsets

FPGrowth was introduced by Han et al. in [HPY00] to find frequent itemsets in a more efficient manner than previous algorithms such as the Apriori algorithm [AS94] or the TreeProjection [AAP00]. We first use the FPGrowth algorithm for selecting the feature names in the feature mining phase of our process (for an explanation of this algorithm see Section 2.4.2.1). The problem with this algorithm is that it utilizes only a single minimum support threshold for all the itemsets. If this threshold is not reached, the itemset will not be kept as a frequent itemset. On one hand, if this threshold is set too low, we risk getting *false positives* (i.e., frequent itemsets that are not an actual representation of co-occurrences with the features). On the other hand, if the threshold is set too high we risk getting *false negatives* (i.e., actual co-occurrences within the features which do not appear in the frequent itemsets).

Given the fact that the features in our dataset are not distributed equally among the products, it is very likely that some features occur significantly more frequently than others. For this reason, using the FPGrowth algorithm for finding itemsets caused the problems previously mentionned (*false positives* or *false negatives*). Also, some of the most infrequent features did not appear in the itemsets at all and in order to be representative of the domain, we needed to have all the features in the itemsets. For these reasons, we switched to the *CFPGrowth* algorithm introduced by Liu et al. in [LHM99] for "*mining association rules with multiple minimum supports*". Unlike the simple FPGrowth algorithm which requires to define one threshold for all the features, the *CFPGrowth* algorithm allows to specify a different minimum support threshold for every single feature. Therefore, it is possible to tweak this threshold for each feature until they all appear in the frequent itemsets. The threshold for each feature can be automatically defined

depending on its support in the dataset.

The way the *CFPGrowth* algorithm works is similar to the FPGrowth algorithm, but due to the introduction of a different minimum support threshold for every feature, the comparison between the support of an itemset and this threshold is different. To determine whether an itemset is frequent or not, the algorithm will compare its support with the minimum of the support thresholds set for the features within that itemset. Let us take the following example:
We have four features, namely A, B, C and D. We have chosen a minimum support threshold of respectively 5, 15, 30 and 40%. Let us say the algorithm found the following itemsets: B, C with a support of 13% and A, B, C, D with a support of 8%. Only the last itemset is considered frequent, even though its support is only 8%, because the algorithm compares the support of that itemset with the minimum support threshold of the items (i.e., 5% for item A).

6.1.2 Binary Association Rules Creation

In order to create binary association rules using the frequent itemsets that have previously been found, we use the algorithm introduced by Agrawal et al. [AIS93] (see Section 2.4.2).

6.1.3 Dealing with an Incomplete Dataset

The definition of a minimum support threshold to mine the association rules allows us to deal with the incompleteness of the dataset. In [CSW08], Czarnecki et al. introduced PFMs. The authors explain the construction of an FM by mining association rules from a dataset of configurations. They retain only association rules with a *confidence* of 100% for the feature hierarchy (*hard constraints*) and those with a *confidence* under 100% but above a certain threshold are kept as *soft constraints* and annotate the FM. In the case of the dataset extracted from Softpedia, we could not find any association rules with a *confidence* of 100%. The reason is that the antivirus dataset we have been working with is incomplete, as explained in Section 6.1. The product-by-feature matrix for the antivirus category contains 165 products and 80 features with an average of 6.5 features per product. Some of these features appear in as few as five products. Therefore we decided to set a confidence threshold above which the association rules were kept either in the FM hierarchy or as CTCs (as we will see in Chapter 7).

6.1.4 Exclusion Clauses

Another problem that we encountered due to the incompleteness of the data is the impossibility to mine exclusion clauses. An exclusion clause has the following form: $f_1 \rightarrow \neg f_2$ and is generally present in an FM. It indicates that if a feature f_1 is present in a product

configuration then feature f_2 should not be present. We could not assume the validity of any exclusion clauses because if the dataset shows that a product does not have a given feature (in the product-by-feature matrix), it does not mean that this product does not in fact have this feature. The absence of a feature in a configuration from the dataset means that this feature was not included in the textual product description on the website from which the list of features has been mined (see Chapter 5 for a detailed explanation of the feature mining phase). In other words, the dataset contains many false negatives. Therefore we do not include the mining of exclusion clauses in our procedure.

The more the data are complete and accurate, the more we can mine exclusion clauses with confidence. This means that while we did not look for any exclusion clause for the antivirus dataset due to its incompleteness, exclusion clause mining could be turned on for a more complete dataset.

6.2 Implication Graph Creation

We can represent the set of the mined association rules as an IG. An IG is a directed graph in which the vertices represent features and where two vertices are connected by an edge if the two features appear in one of the mined association rule : the predecessor vertex corresponds to the antecedent feature of the association rule and the successor vertex corresponds to the consequent feature of the association rule. In other words, in an IG, vertex f_1 is connected to vertex f_2 if an association rule $f_1 \rightarrow f_2$ has been mined.

In our context, the edges in IG are weighted. The weight on the edge between any two given features f_1 and f_2 is the confidence of the association rule $f_1 \rightarrow f_2$. Therefore, the IG represents the probabilistic dependencies between features. Figure 6.2 shows an example of an IG for the sample antivirus related features from Table 6.1. The weights on the edges will be useful in Chapter 7 for the computation of a tree structure for the FD.

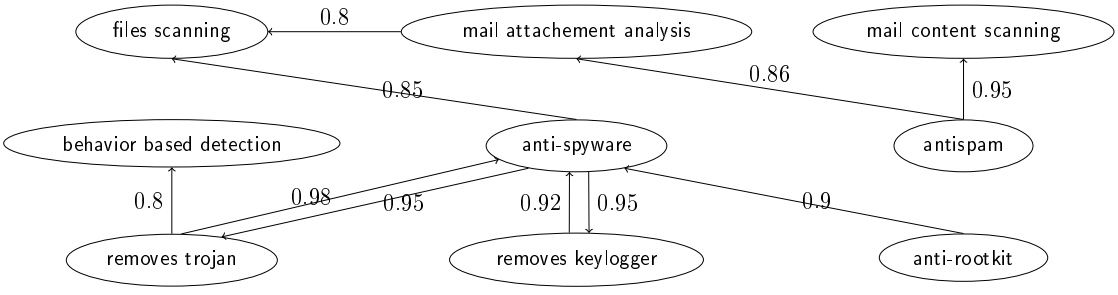


Figure 6.2: Implication Graph

6.3 Strongly Connected Components Detection

In this section, we first discuss the detection of SCCs in an IG to discover AND-groups and then address the limitation of this technique when applied to an IG made of probabilistic association rules.

In a directed graph G, an SCC is a subset of vertices in G for which there exists a path from each vertex to every other vertex. In the IG, an edge from vertex v_1 to vertex v_2 means that the presence in a configuration of feature f_1 represented by vertex v_1 implies the presence of feature f_2 represented by vertex v_2. Therefore, an SCC in an IG mined from a dataset of product configurations represents a set of features for which the presence of any of its feature elements in a configuration implies the presence of all the other features from the set. This comes from the fact that the implication between features presence in configurations is transitive and that there exists a path from every vertex to any other vertex of the SCC. We can say that any pair of features belonging to a common SCC are bi-implied.

The *transitive closure* $C(G)$ of a graph G is a graph such that there is an edge connecting any pair of vertices that are connected by a path in G :

$$\forall (v,w) \in C(G), \exists_{0 \leq i < N} \{(x_i, x_{i+1})\} \subseteq G \, : \, x_0 = v \, and \, x_N = w$$

A new representation that preserves the transitive closure can be given to the IG G by merging vertices that represent bi-implied features as one vertex. Because an SCC is logically equivalent to a set of bi-implications between its features, all vertices belonging to an SCC can be merged as one vertex without loosing any information. Let us call G' the graph derived from G in which each SCC of G has been merged as one vertex. G' is the same graph as G except that each SCC from G has been replaced by a vertex representing the conjunction of its features (see IG on the right in Figure 6.3). For each edge in G going from a vertex v outside an SCC C to a vertex inside C, there exists an edge in G' going from v to the vertex representing C. Respectively, for each edge in G going from a vertex inside SCC to a vertex v outside SCC, there exists an edge in G' going from the vertex representing C to v. G and G' have the same transitive closure. This is illustrated in figure 6.3.

In [ACSW12] Andersen et al. compute SCCs in IGs to detect AND-groups. The mapping from SCCs to AND-groups is motivated by the fact that the implications between the features in the IG are transitive. Because there is a path between any two features in a SCC, each presence of a feature of an SCC in a configuration implies the presence of every other feature of the SCC in this configuration. However, an IG such as the one shown in Figure 6.2 is not made of implications but of probabilistic association rules which are not transitive. It follows that features that do not occur often together in configurations may be considered as parts of the same AND-group. So, in case of an IG using probabilistic association rules, AND-groups can be computed from SCCs to offer the user an approximation of potential AND-groups but these groups must be

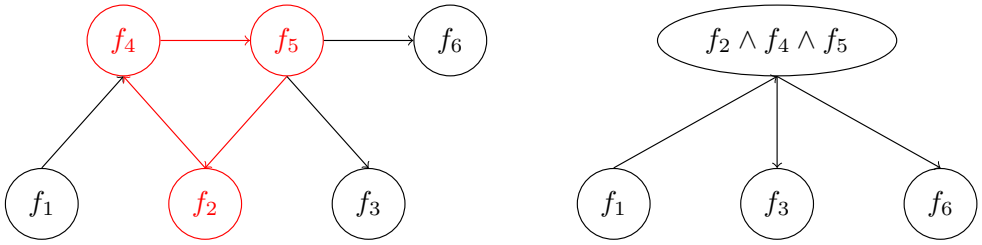


Figure 6.3: The nodes f_2, f_4 and f_5 form an SCC in the IG on the left and are merged as one node in IG on the right. The IG on the right has the same transitive closure as the IG on the left

Table 6.2: Example dataset

Feature1	Feature2	Feature3
X	X	X
X	X	X
X	X	X
X	X	
X		

manually reviewed at the end of the process. Figure 6.4 shows a simplified example of an IG mined from the dataset represented in 6.2. The threshold for the mining of the association rules is 0.75. In the resulting IG, $feature_1$, $feature_2$ and $feature_3$ form an SCC. Therefore, we could suggest that the three features form an AND-group. However, by doing so, $feature_1$ and $feature_3$ appear in the same AND-group while $support(feature_1 \rightarrow feature_2) = 0.6$ which is lower than the threshold (0.75) initially defined to mine association rules between features. In other words, the two features are in a common AND-group while they do not frequently appear together in configurations regarding the predefined association rules minimum support threshold.

In our procedure, we do not directly look for AND-groups in the overall IG but we detect the SCCs in sub-graphs of the IG. This is because we divide the IG in multiple sub-graphs while computing a hierarchy between the features of the sub-graphs. Chapter 7 discusses the extraction of an FD from the IG and the

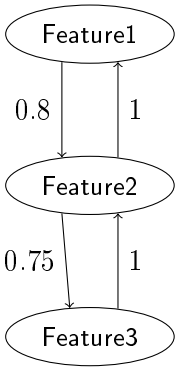


Figure 6.4: An AND-group made of the three features can be suggested as they appear in the same SCC. However, $feature_3$ is only present in 60% of the configurations featuring $feature_1$

6.4 Minimal Disjunctive Rules Mining

In this section, we discuss the mining of minimal disjunctive rules to identify OR-groups. Firstly, we discuss the mining of minimal disjunctive rules for complete dataset by computing prime implicates from a DNF formula encoding the variant products configurations expressed in the dataset. Secondly, we discuss the solution for the mining of disjunctive rules from an incomplete dataset that we use in our procedure.

6.4.1 Mining Minimal Disjunctive Rules from a Complete Dataset

A **disjunctive rule** $f_1 \rightarrow f_2 \vee f_3 \vee \cdots \vee f_n$, represents an implication from a single literal to a *clause* (i.e., a disjunction of literals).

The disjunctive rules are found by identifying *prime implicates* among features. A clause C is an *implicate* of the logical expression e, if and only if C is not a tautology and $e \Rightarrow C$ is a tautology. An implicate C of expression e is said to be *prime* if and only if there is no clause C' obtained by removing any literal from C such that C' is an implicate of e (i.e., C is minimal).

An implicant C of a logical expression e is a term such that $C \Rightarrow e$ is a tautology. An implicant C is said to be *prime* if it is minimal.

By observing whether disjunctions of features are prime implicates, we can identify OR-groups of minimal size among features. For each mined disjunctive rule $f_1 \rightarrow f_2 \vee f_3 \vee \cdots \vee f_n$, if f_1 is the parent feature of all features $f_2, f_3, \cdots, f_n$ in the hierarchy of the FD, then the disjunctive rule can be graphically represented as an OR-group.

The computation of the prime implicates is the most difficult step of the algorithm. In [ACSW12], Andersen et al. discuss the complexity of computing prime implicates. The

authors underline the fact that if π is a prime implicate of φ, then $\neg\pi$ is a prime implicant of $\neg\varphi$ so that the computation of the prime implicates for a formula φ in DNF can be reduced to the computation of the prime implicants of $\neg\varphi$ which is in CNF. Finding prime implicates for a CNF formula is an NP-complete problem ([ACSW12]).

Finding OR-groups from a formula consists in finding prime implicates : for a formula φ over variables $f_1, \cdots, f_n$, an OR-group of a feature f in φ corresponds to a prime implicate $(f_1 \vee \cdots \vee f_k)$ of $\varphi \wedge f$ corresponding to the children of f in the IG. By negation of this implication, it follows that an OR-group corresponds to a prime implicant $(\neg f_1 \wedge \cdots \wedge f_k)$ of $\neg\varphi \vee \neg f$.

The product-by-feature matrix can be translated into a DNF formula φ expressing the list of existing variants of the product configurations. To discover the prime implicates of a DNF formula φ, Andersen et al. [ACSW12] propose to find the prime implicants of $\neg\varphi$ which is in CNF by reducing the problem to a Binary Integer Programming (BIP) problem ([VMMO], [Sil97]). BIP is NP-complete ([GCI79]) and [VMMO] propose two SAT-based algorithms to solve BIP.

If we are only interested in disjunctive rules that can be graphically represented as OR-groups in the final FD (i.e., rules for which the antecedent feature is a parent of all the consequent features in the hierarchy of the FD), the computation of these rules must be done after the elicitation of the FD from the IG and, when solving the BIP to find prime implicants $(\neg f_1 \wedge \cdots \wedge f_k)$ of $\neg\varphi \vee \neg f$, we can remove all variables in the BIP corresponding to non-children of f.

If we are interested in discovering all the minimal disjunctive rules, no variables are removed from the BIP and the mining of the rules can be done before the elicitation of the FD. In that case, once the FD has been discovered, the disjunctive rules that can not be graphically represented in it are kept aside as CTCs.

6.4.2 Mining Minimal Disjunctive Rules from an Incomplete Dataset

In our procedure, we implemented a solution to mine minimal disjunctive rules that copes with incomplete dataset. We allow the user to define a minimum threshold for the confidence of the disjunctive rules. Therefore, the algorithm may suggest potential disjunctive rules despite the disparity of the product-by-feature matrix.

When the scope of the disjunctive rules mining is reduced to parent features in the FD and their respective sets of children, it becomes feasible to compute prime implicates in a brute-force manner - i.e., by counting co-occurrences between the parent feature and disjunctions of its children in the dataset in order to find minimal disjunctive rules for which the confidence is above the predefined threshold. Therefore, we apply the mining of the minimal disjunctive rules after the elicitation of the FD (see Section 7.6).

Chapter 7

Structuring the Feature Model

In this chapter we present a procedure that extracts an FM from an IG. While the previous chapter was about discovering the logical formula to be represented by the FM, this chapter focuses on finding an FM which is a good presentation of the formula.

The creation of the FM is based on two approaches :

1. Extract a tree from the input IG which is a good hierarchical representation of the features.
2. Use text-mining techniques to enhance the readability of the FM and reduce its cognitive complexity. Two techniques are used : clustering features that deal with the same aspect of the domain so that they will be grouped together in the FM (1) and creating additional abstract nodes (2).

The chapter begins by introducing this part of the process and gives an overview in Section 7.1. Then Section 7.2 explains how the features are clustered. Section 7.3 describes how to find the structure between features that are in the same cluster. Then Section 7.3.2 explains how the abstract nodes are included in the FD. Section 7.4 describes how to find a structure for the final FD. Finally, Section 7.6 shows how to recover the CTCs to obtain the final FM.

7.1 Overview

In the previous chapter, we have successfully mined an IG from the product-by-feature matrix using data mining techniques. We have explained how to discover AND-groups by looking for SCCs in the IG and how to discover OR-groups by mining disjunctive clauses. However, the IG is not of much use for a domain analyst as the only information it represents is whether features should or should not be selected given a set of already selected features (i.e., it can help derive a partial configuration into a complete configuration). In order to deliver all the relevant information about the product configurations to a domain analyst, we need a more convenient structure. We need an FD which gives a readable and well-structured overview of the domain.

To extract an FD from the IG, we first tried to compute a tree from the IG by applying the Chu-Liu/Edmonds optimum branching algorithm. This is a minimum spanning tree algorithm for directed graphs. We applied it to the IG, the edges of which are weighted by conditional probabilities between features, to find its maximum tree. The Chu-Liu/Edmonds optimum branching algorithm is explained in details in Section 7.3.1. The problem we encountered with this approach is that, due to the sparsity and the incompleteness of the product-by-feature matrix, the tree computed for the IG was very flat (lots of features were connected to the root). We decided to use the *feature descriptors* as additional input to compensate for the sparsity of the matrix and text-mining techniques to enhance the structure of the FD.

Section 7.2 describes how we use feature descriptors to cluster features so that features dealing with the same aspect of the domain will end up grouped together in the FD. Once the features have been separated into clusters, the tree structure of the FD can be computed in two steps. During the first step, a FD is computed for each of the clusters (see Section 7.3). Secondly, all the FDs are connected together to form an overall FD (see Section 7.4). Once the FD has been found, the CTCs can be recovered (see Section 7.6) and the FD along with the CTCs form the final FM. Figure 7.1 shows an overview of the FM structuring phase.

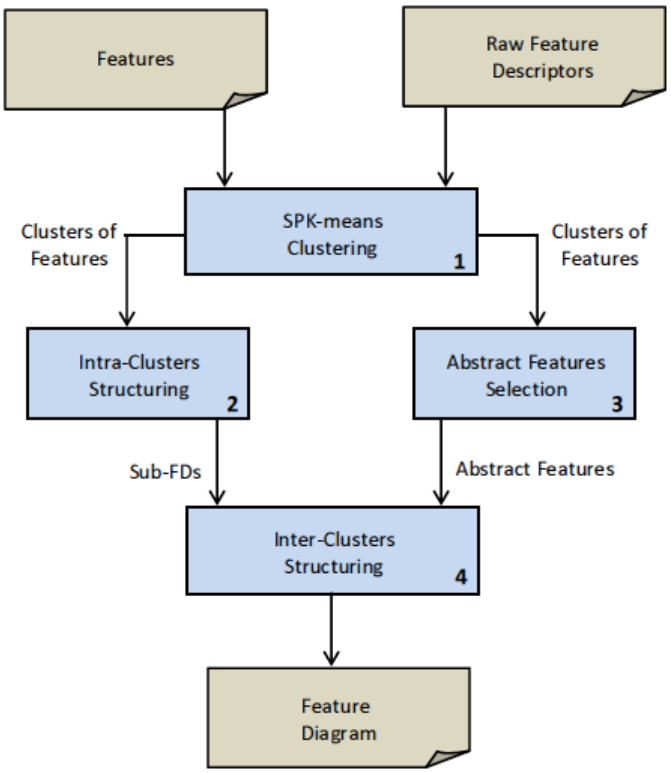


Figure 7.1: FD structuring phase overview.

7.2 Clustering Features

In a user study that we ran (see Section 9.2), the users were asked to create an FM given a set of features found by our feature mining process and were separated into two different groups. Both groups of users started their tasks by separating the features into groups. They were trying to group features that covered the same aspects of the domain together. We base the computation of the FD structure on the assumption that it is regular for practitioners to partition features into groups, to look for a structure inside each one of the groups and finally to connect them together in order to form the final FM. We try to automate this trend in our process by clustering the features that share a common topic together using the SPKMeans algorithm.

The features are separated into N clusters based on the terms used in their associated *feature descriptors*. The approach used to find the right number of clusters (N) is the same as the one used in [DGH+11]. Once those clusters have been found, we extract the smallest sub-graph of the IG for every cluster that contains only the features of that cluster. If there was a directed edge between two features of different clusters, it will be added to the CTCs.

Table 7.1: Features separated in three clusters

spyware, protection	mail, spam	scan, detection, files
anti-spyware removes keylogger removes trojan anti-rootkit	mail content scanning mail attachement analysis antispam	files scanning X behavior based detection

If we go back to our example (see Figure 6.2 on page 45), we can see that the IG contains nine features. By running the SPKMeans algorithm on this set of features, we separate them into three clusters as shown in Table 7.1. The resulting three sub-graphs (extracted from the IG in Figure 6.2) are pictured in Figure 7.2. The edges that were in the IG and which are not in those clusters will be added to the CTCs (see Section 7.6.2). In this example, those edges are: $antispyware \rightarrow files\,scanning$, $mail\,attachment\,analysis \rightarrow file\,scanning$ and $removes\,trojan \rightarrow behavior\,based\,detection$.

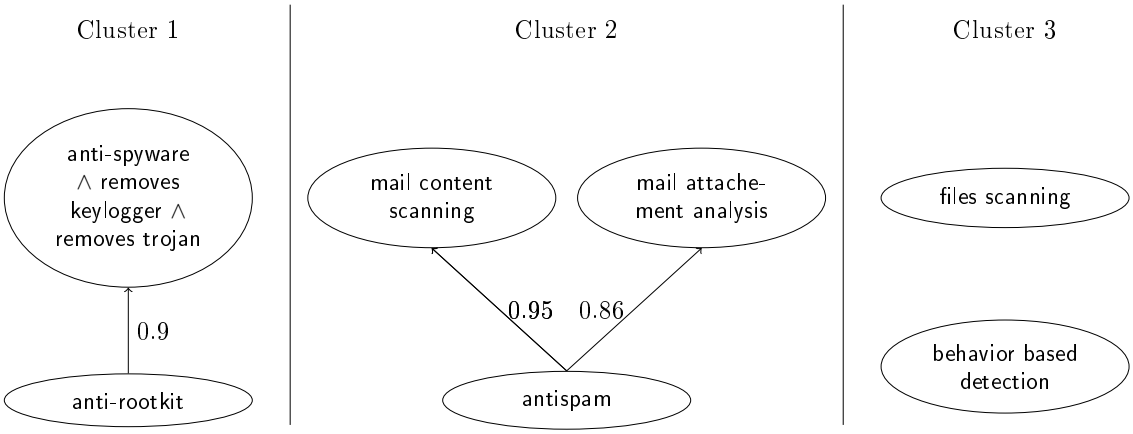


Figure 7.2: IG divided in subgraphs after clustering

7.3 Intra-Clusters Structure

Now that the IG has been divided into sub-graphs, we need to find a tree structure for each of these sub-graphs as they will later be merged to form the final FD.
Once an optimum branching (tree structure) has been found for each of the subgraphs, *abstract features* are added. The idea is to add an abstract feature as root node for each subtree. At this stage, we now have several sub-FDs that need to be connected to form the overall FD. The computation of the structure between the sub-FDs is explained in Section 7.4.

7.3.1 The Chu-Liu/Edmonds Optimum Branching Algorithm

Chu and Liu, Edmonds, and Bock have independently devised an algorithm to compute an *optimum branching* in a directed graph. The Chu-Liu and Edmonds algorithms are identical. The Bock algorithm is similar but relies on the use of matrices instead of graphs. [Tar77] gives an implementation of the Chu-Liu/Edmonds algorithm.

The selection of an FD from an IG can be solved as a *minimum spanning tree* problem. We use the Chu-Liu/Edmonds algorithm in order to find the best FD contained in an IG. In our case, edges in the directed graph are weighted with conditional probabilities between features and the algorithm discovers the tree that maximizes the product of these probabilities.

We transcribed the implementation described in [Tar77]. The pseudo-code for this algorithm is provided below (see Algorithm 2 [Tar77]) and the code for our implementation can be found in Section B.1 on page 103. The algorithm runs in $\mathcal{O}(m.log(n))$ where n is the number of vertices in the graph and m the number of edges. Here are seven operations that are used in the algorithm:

- FIND(x) returns the name of the set containing element x;
- UNION(A,B) adds the elements of set A to set B, destroying set B;
- QUNION(C,D) adds the elements in queue D to queue C;
- MAX(C) returns the largest element in queue C, deleting this element from the queue;
- INIT(C,L) initializes a queue C to contain all elements in the list L;
- ADD(a,C) adds a constant a to the value of all elements in queue C;
- ENTER(i) gives the unique edge in H entering SCC component i (if there is no such entering edge, ENTER(i) = (0,0)).

Algorithm 2 Chu-Liu/Edmonds Optimum Branching Algorithm

```
1: roots ← ∅
2: for i ← 1 to n do                          ▷ n is the number of nodes in the graph
3:     INIT(i, I(i))
4:     initialize an S-set named i containing i as its only element
5:     initialize a W-set named i containing i as its only element
6:     ENTER(i) ← (0, 0)
7:     roots ← roots ∪ {i}
8:     min(i) ← i
9: H ← ∅
10: rset ← ∅  ▷ The set rset gives the root components of G(H) with no entering edges
    of positive value
11: while roots ≠ ∅ do                        ▷ n is the number of nodes in the graph
12:     delete some entry k from roots
13:     (i, j) ← MAX(k)
14:     if v(i, j) ≤ 0 then
15:         rset ← rset ∪ {k}
16:     else
17:         if SFIND(i) = k then
18:             roots ← roots ∪ {k}
19:         else
20:             H ← H ∪ {(i, j)}
21:             if WFIND(i) ≠ WFIND(j) then
22:                 WUNION(WFIND(i), WFIND(j))
23:                 ENTER(k) ← (i, j)
24:             else
25:                 val ← ∞
26:                 (x, y) ← (i, j)
27:                 while (x, y) ≠ (0, 0) do
28:                     if c(x, y) < val then
29:                         val ← c(x, y)
30:                         vertex ← SFIND(y)
31:                     (x, y) ← ENTER(SFIND(x))
32:                 ADD(val − c(i, j), k)
33:                 min(k) ← min(vertex)
34:                 (x, y) ← ENTER(SFIND(i))
35:                 while (x, y) ≠ (0, 0) do
36:                     ADD(val − c(x, y), SFIND(y))
37:                     QUNION(k, SFIND(y))
38:                     SUNION(k, SFIND(y))
39:                     (x, y) ← ENTER(SFIND(x))
40:                 roots ← roots ∪ {k}
```

7.3.2 Abstract Features

The next step in the FM elicitation phase consists in automatically creating abstract features and position them in the FD in construction to enhance its readability.

FMs that are manually built by human experts differ from automatically built FMs partly because experts typically add *abstract* features in the structure of the FDs in order to reduce their cognitive complexity. An *abstract* feature in an FM, contrary to a *concrete* feature, is a node that does not directly represent a specific and tangible feature. It has been added to group features in the hierarchy that are related to a common aspect of the modeled domain. The creation of abstract features results in an additional level in the hierarchy and aims to enhance the readability of an FM. Figure 7.3 gives an example of an abstraction node with the additional node *Gear*. It shows that all features related to the concept of *gear* are isolated together and that the FM becomes less flat. These types of changes can help enhance the clarity and the expressiveness of an FM, especially when the number of features becomes important ([BG11] discusses the use of structural metrics to evaluate the maintainability of FMs).

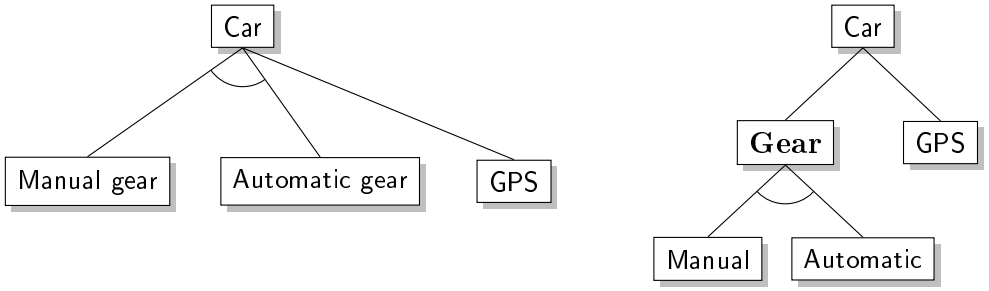


Figure 7.3: The FMs are equivalent representations. The abstract feature *Gear* has been added.

We decided to add an abstract feature as root for every sub-tree discovered previously in Section 7.3. To find the name of the abstract nodes, we use the results from computations that have been made by the SPKMeans algorithm when the features were clustered (see Section 7.2). As explained in Section 7.2, the SPKMeans algorithm is based on a vector-space model and attributes weights to terms according to TFIDF statistics. At the end of the last iteration of the SPKMeans algorithm, the terms related to a cluster that were attributed the highest weights are the terms that reached the highest ratio between two elements : they are the terms that are the most redundant in the feature descriptors of the cluster and that are the most specific to these descriptors (i.e., they appear more repeatedly in these descriptors than in descriptors of other clusters). In other words, the algorithm has recognized these terms as the most representative terms of the features in the cluster. This is why we use the conjunction of these terms to form the name of the abstract feature of the cluster. The abstract feature found for a cluster is added as a root for the sub-tree corresponding to this cluster.

Figure 7.4 shows the sub-trees computed for the sub-graphs in Figure 7.2 and their added abstract feature roots.

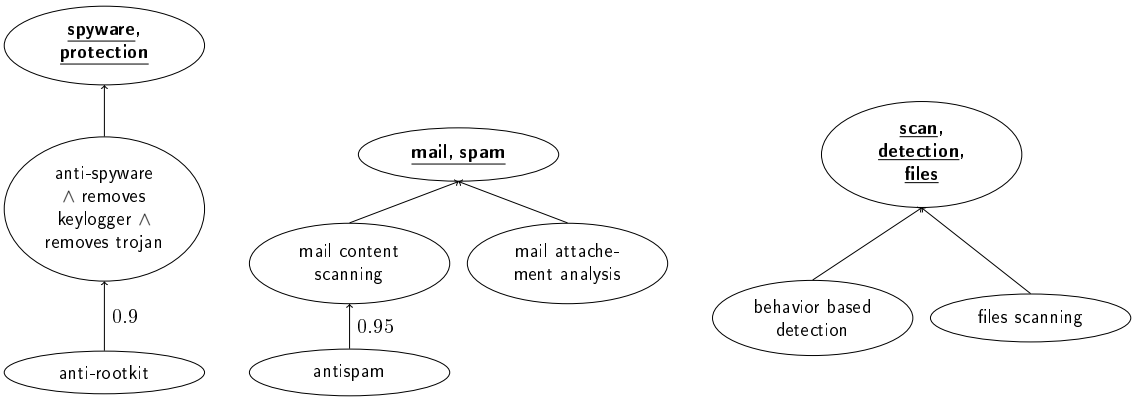


Figure 7.4: For each sub-graph, a tree-structure is computed and an abstract feature is added as root.

7.3.3 AND-Groups

In Section 6.3, we saw how we could detect SCCs to identify features that can appear together in an AND-group. We apply the SCC detection to each sub-graphs found in Section 7.2. The features that belong to a common SCC can be merged into one node that now represent a conjunction of features as shown in Figure7.4 for the features *anti-spyware*, *removes keylogger* and *removes trojan*.

The features from the disjunction can be represented as an AND-group in the FD. To do so, the merged node has to be unfolded. Unfolding the merged node can be done by selecting one of the feature as the parent of the AND-group and all the other features as its children. Given an AND-group, we choose the parent feature as the feature that maximizes the minimal co-occurrence with other features of the group in the product-by-feature matrix. Formally, the parent feature f maximizes the number of configurations involving f and any other feature of the AND-groups :

$$\arg\max_{f}(min\{P(f_i|f)|f \neq f_i\}) \tag{7.1}$$

For example, and as shown in Figure 7.2 and Figure 7.8, the feature *anti-spyware* is selected as parent of the group.

7.4 Inter-Clusters Structure

Now that a tree structure has been found for each sub-graph, the sub-graphs need to be integrated into one overall FD. We call the structure between the sub-trees the *inter-clusters structure*. The inter-clusters structure is found in two steps. The first step consists in connecting the FDs (found in Section 7.3) into the integrated FD the same way it was done for the IG in Section 6.2. The second step consists in finding a tree structure from the inter-clusters structure.

In the first step, instead of connecting individual features, we connect the sub-trees into a graph of sub-trees. To discover the edges connecting the sub-trees, we run the association rules mining algorithm that we had previously used to find edges for the IG in Section 6.2. However, the dataset used as input of the association rules algorithm is changed. The features in the product-by-feature matrix are replaced by their corresponding clusters label before mining the association rules. In other words, the association rules mining is not processed on the product-by-feature matrix but on the *product-by-cluster* matrix. In the product-by-cluster matrix, the lines represent products (as in the product-by-feature matrix) and the columns represent clusters of features. If the cell (i, j) is filled with a 1, it means that the product p_j possesses at least one of the feature in cluster c_j. Conversely, if it is filled with a 0, it means that p_j possesses none of the features from cluster c_i. Figure 7.5 shows how the product-by-cluster matrix is built from the product-by-feature matrix and the list of clusters.

	f_1	f_2	f_3	f_4	f_5	f_6	f_7
p_1	1	0	1	0	1	0	0
p_2	0	1	1	0	0	1	1
p_3	1	0	1	0	1	1	0
p_4	0	0	0	1	0	1	1

$c_1 = \{f_1, f_2, f_3\}$
$c_2 = \{f_4, f_5\}$
$c_3 = \{f_6, f_7\}$

	c_1	c_2	c_3
p_1	1	1	0
p_2	1	0	1
p_3	1	1	1
p_4	0	1	1

Figure 7.5: Construction of a product-per-cluster matrix (right) from a product-per-feature matrix (left) and a list of clusters of features (middle).

In Section 7.3.2, we made sure that for every cluster, the corresponding FD had a root (the added abstract feature). Therefore, the mined association rules between the clusters can be represented as edges connecting the roots of the FDs as shown in Figure 7.6 where the descendant concrete features of the abstract features have been *folded* into their ancestor abstract feature.

The second step consists in finding a tree structure for the overall FD. This is done by following the same steps as described in Section 7.3. The Chiu-Liu/Edmonds algorithm is applied to find the optimal branching between the abstract nodes.
Figure 7.7 shows the resulting FD after applying the different steps of our procedure. The FD is the same as in Figure 7.6 but with the concrete features unfolded under the abstract features.

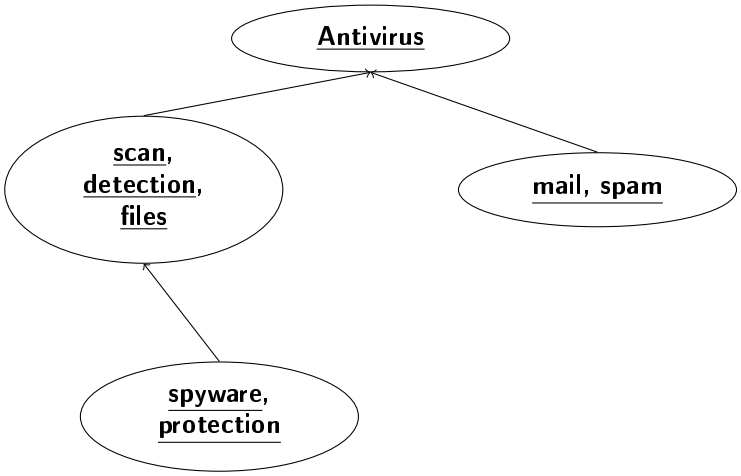


Figure 7.6: Representing the inter-clusters structure comes down to connecting the roots of the sub-trees. An abstract feature *Antivirus* is added as root of the entire FD

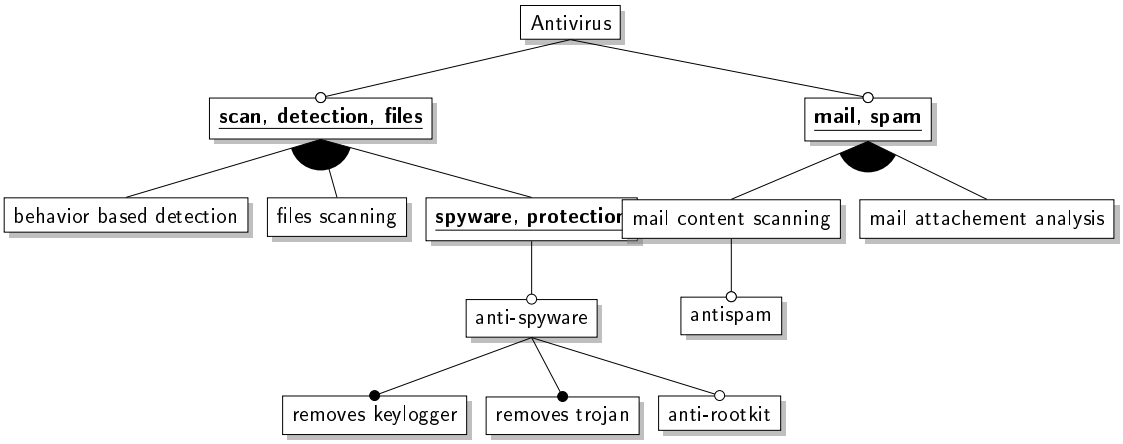


Figure 7.7: Resulting FD

7.5 Motivations behind Techniques Used in the Structuring of the Feature Diagram

The motivations behind the techniques used to structure the FD are the following :

1. **Cluster features** The idea is to mimic the users' behaviour of separating the features into groups so that the features dealing with the same aspect of the domain will be gathered on the same branch of the FD.

2. **Apply the Chu-Liu/Edmonds algorithm based on features probabilistic dependencies** The hierarchy of features in the FD indicates how features encourage each others presence in product configurations. We use statistical information offered by the product-by-feature matrix to compute the tree-structure that maximizes the product between the conditional probabilities between the features.

3. **Generate abstract features** Additional abstract features are added to the hierarchy in order to enhance its readability. Each abstract feature name indicates representative terms for the successor concrete features. Abstract features can be graphically represented in a different style than concrete features in order to help the user identifying the various parts of the FD an their semantics.

In Section 7.3.1, we saw that the confidence of the mined association rules were used to decide a tree structure for the FD. Instead of using the confidence of the association rules to weight the edges between features, we could use another measure such as the *cosine similarity* measure based on the *raw feature descriptors* of the features. The weight could also depend on both association rules confidence and *cosine similarity* between *raw feature descriptors*. In this case, the decision of the tree structure would depend on both measures. Because we already clustered the features in the FD structure based on the *cosine similarity* between *raw feature descriptors*, we decided to only use the information given by the product-by-feature matrix to weight the edges and we used the confidence of the association rules. This decision is motivated by the fact that statistics about the co-occurrences between features are more reliable than the *cosine similarity* to determine which feature may imply another feature. Indeed, the *cosine similarity* by itself does not indicate the nature of the relationship between two features. Two features with a high *cosine similarity* could imply each other's presence in a configuration as they may cover a same aspect of the domain. However, they could also exclude each other's presence in a configuration if they fulfil the same functionalities and therefore make each other obsolete.

In [CSW08], Czarnecki et al. recommend the respect of the *maximality property* for FMs that have been automatically extracted from logical formulas. This property states that "*the resulting feature model graphically exposes maximum of logical structure*". To satisfy this property in our context, we should try to find the FD that exposes as many association rules as possible by prioritizing the rules with the highest confidence. The procedure we describe in Section 7.3 and Section 7.4 does not respect the maximality

priority because we decide not to represent association rules between features that do not belong to the same cluster (these rules are recovered as CTCs, see Section 7.6.2). However, if we decide not to process the clustering phase and immediately compute a FD for all the features with the Chu-Liu/Edmonds algorithm, the resulting FM presents two problems. Firstly, different features covering a same aspect of the domain may be positioned far apart from each other in the resulting FD. This would make the FM harder to understand for users. Secondly, the computation of the structure of the FD would exclusively rely on the product-by-feature matrix which may contain errors and be incomplete as it has been automatically mined from incomplete textual descriptions of products. By clustering the features and not only using the statistical information of the matrix, we can use the *raw feature descriptors* as an additional input to decide the tree structure of the FD and to incorporate tactics to enhance the maintainability of the FM such as grouping related features or adding abstract features. In case this approach discards a lot of association rules (i.e. association rules recovered as CTCs), the CTCs with the highest confidence may be represented in the FD with doted edges while the CTCs with the lowest confidence may be represented with text aside the FD.

To summarize, we use both statistical and text-mining techniques to structure the FD. We believe that clustering the features in the FD is complementary to the statistical analysis of the product-by-feature matrix and increases the readability of the FD. The trade-off for this cluster-based approach is that it does not necessary respects the maximality property. Therefore, a lot of association rules may be recovered as CTCs (see Section 7.6.2 for a description of CTCs recovery).

7.6 From Feature Diagram to Feature Model

An FM is formed by an FD and possibly some CTCs. Now that the FD has been extracted from the IG, the CTCs can be recovered.

7.6.1 Disjunctive Rules Recovery

Section 6.4 explains how to mine minimal disjunctive rules from the product-by-feature matrix. A disjunctive rule of the form $f_1 \rightarrow f_2 \vee f_3 \vee \cdots \vee f_n$ where $f_2 \vee f_3 \vee \cdots \vee f_n$ is a prime implicate of f_1 and where feature f_1 is a parent of features $f_2, f_3, \cdots, f_n$ in the FD can be graphically represented as an OR-group in the FD. Considering the size of the dataset we have been working with, it is not feasible to look for all minimal disjunctive rules among all features and then filter them. However, mining the prime implicates becomes feasible when its scope is reduced to disjunctive rules $f_1 \rightarrow f_2 \vee f_3 \vee \cdots \vee f_n$ for which feature f_1 is a parent of features $f_2, f_3, \cdots, f_n$ in the FD. By doing so, we only look for disjunctive rules that can be graphically represented. In case all minimal disjunctive rules are mined, some of them can be graphically represented as OR-groups but the other has to be represented as CTCs aside the FD.

7.6.2 Association Rules Recovery

To find the CTCs that must be represented aside the FD, we need to recover the association rules from the IG that have been discarded by the clustering of the features or that have not been selected for the FD by the Chiu-Liu/Edmonds algorithm. Figure 7.8 shows the FD annotated by CTCs and forming the final FM for the sample antivirus features. Now, each association rule shown in Figure 6.2 is represented in the FM in Figure 7.8 either as one edge of the FD hierarchy or as one CTC.

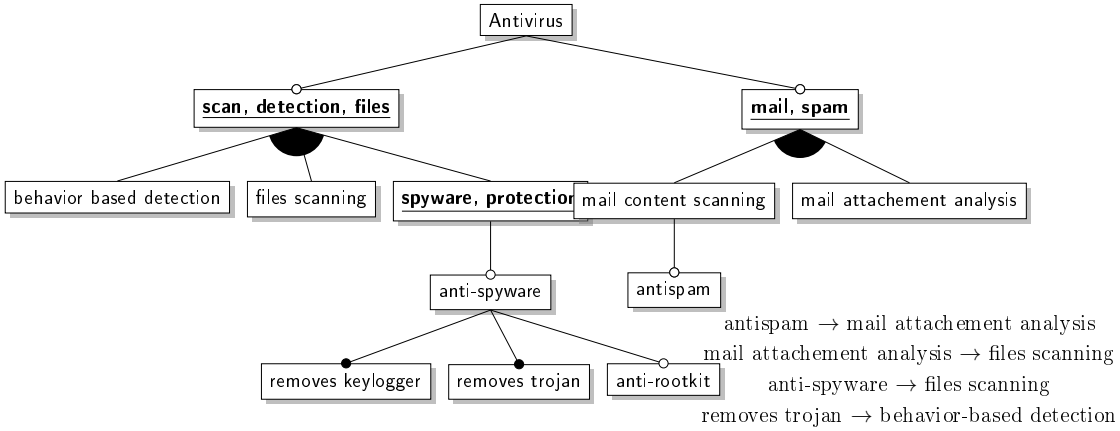


Figure 7.8: Resulting FM

Chapter 8

Tool

In this chapter, we present the tool that we have developed, which implements the extraction procedure that we have described from Chapters 4 to 7. We describe the architecture of the tool in Section 8.1. Finally in Section 8.2 we describe how we have implemented it and we give some metrics regarding its complexity.

8.1 Architecture

Figure 8.1 depicts the different components of the tool, and the interfaces between them. All the components are meant to provide a specific interface to another component. We have tried to make the tool as modular as possible so that we could change the algorithms used easily (e.g., FPGrowth instead of *CFPGrowth*). We have also used a configuration file for all the different parameters of all the algorithms. This means that the extraction procedure can easily be configured using the Graphical User Interface (GUI) (see Section 8.3).

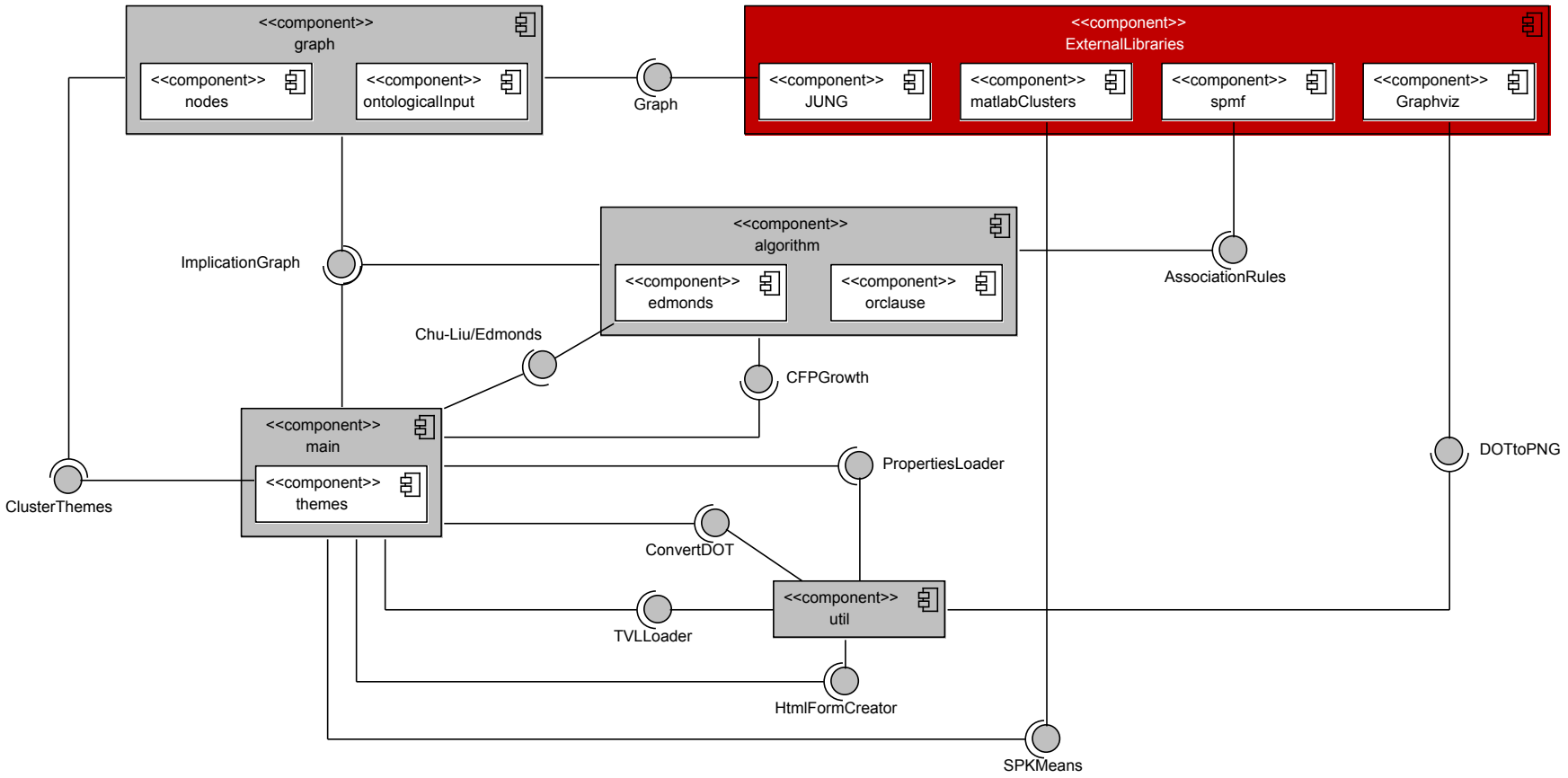


Figure 8.1: Component diagram of the tool implementing our extraction procedure.

8.1.1 Components

In this section we give a brief description of the components depicted in Figure 8.1.

8.1.1.1 *Main*

The *main* component uses all the other components to implement the steps of the extraction procedure that we have previously introduced.

8.1.1.2 Graph

The *graph* component provides an extension of the *SparseMultigraph* class that is part of the JUNG[1] library for Java. We have extended this class to include the specificities of FMs (AND-groups, OR-groups, features) and to provide various methods to merge graphs or convert them to DOT[2], or to Text-based Variability Language (TVL).

8.1.1.3 Algorithm

The *algorithm* component provides three different algorithms: (1) Chu-Liu/Edmonds, (2) OrClauses and (3) CFPGrowth.

We have implemented (1) by following the implementation of the Chu-Liu/Edmonds' algorithm proposed by Tarjan [Tar77], which took us about one day of peer-programming. This implementation can be found in Section B.1 on page 103.

(2) finds the disjunctive rules within the dataset and creates OR-groups within the graph according to those disjunctive rules.

(3) converts the data to a format usable by the CFPGrowth algorithm provided by the Sequential Pattern Mining Framework (SPMF)[3].

8.1.1.4 Util

The *util* component provides various utilities to the *main* component. Those utilities include methods to convert DOT files to PNG, validate product configurations against an FM using TVL, load properties from the configuration file and create HTML forms for the evaluation.

[1] JUNG is a library that provides a language for modeling graphs http://jung.sourceforge.net/.

[2] DOT is a format created by Graphviz to draw directed graphs http://www.graphviz.org/pdf/dotguide.pdf.

[3] SPMF is an open-source data mining library http://www.philippe-fournier-viger.com/spmf/index.php?link=documentation.php

8.1.1.5 ExternalLibraries

The *ExternalLibraries* component represents the external libraries that we use. Those libraries include *JUNG*, *matlabClusters*, *spmf* and *Graphviz*. We have already given a brief explanation of all the libraries except for matlabClusters. This library was developed by some PhD students of DePaul University, Chicago. It provides a set of clustering algorithms implemented in Matlab.

8.2 Implementation

We have spent the entire internship at DePaul University, Chicago, which amounts to five months, developing the tool along with our research work to design the extraction procedure. In terms of hours, working on the implementation alone took a total of about 850 hours. Table 8.1 shows three different metrics: number of lines of code, in thousands (KLOC), number of classes and time spent.

Table 8.1: Metrics regarding the tool implementation.

Metrics	#
KLOC	5.9
Number of Classes	36
Time Spent	850 hours

8.3 Graphical User Interface

The GUI of the tool allows users to configure the different parameters of the extraction procedure, which will be saved in a configuration file, and then execute it. After the tool has finished mining the FM from the product-by-feature matrix, a folder will be created in the user's local files. This folder contains the TVL representation of the FM, the DOT and PNG files which graphically represent the FM and some metrics about it. An HTML form can also automatically be created to evaluate the FM.

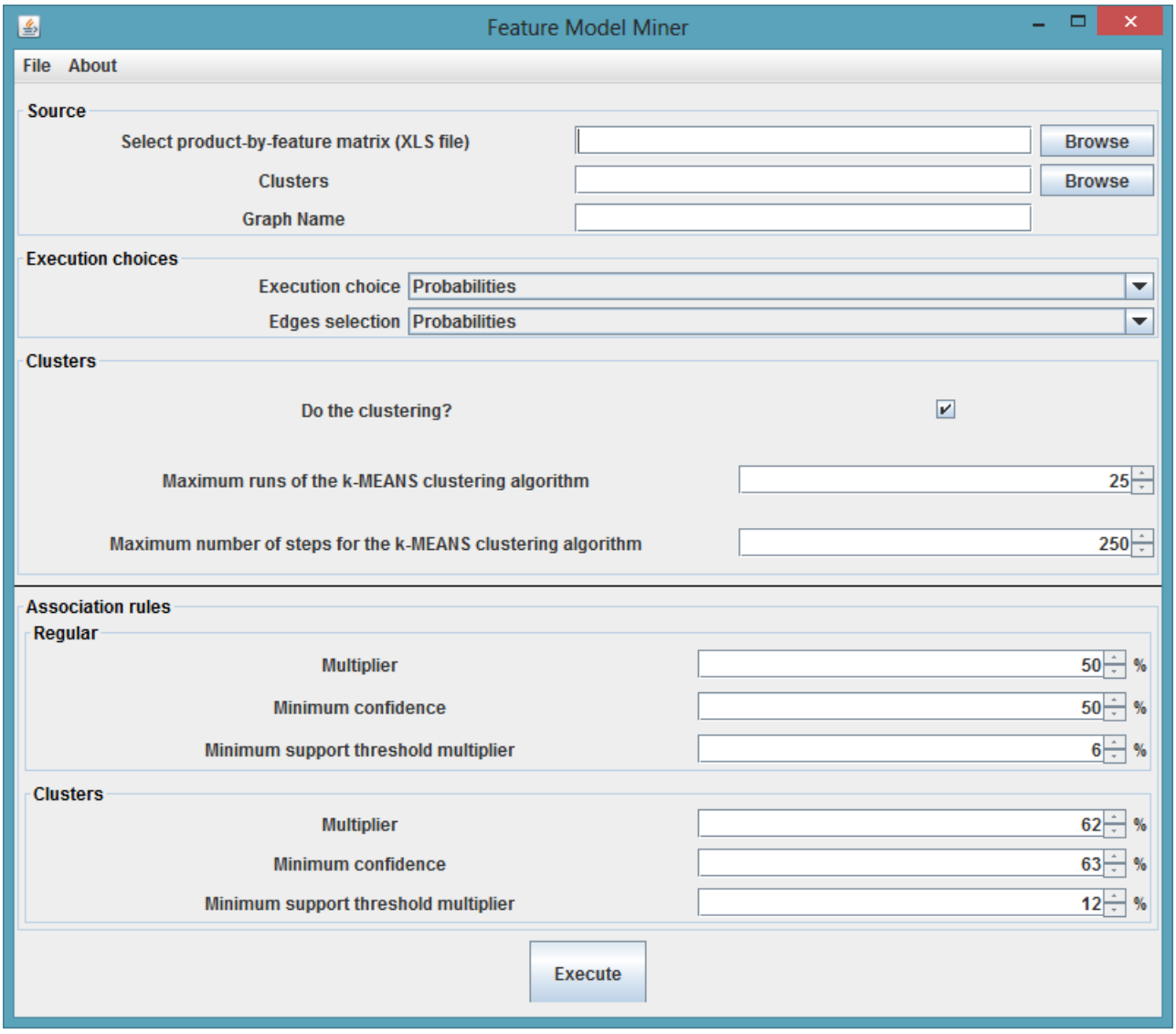


Figure 8.2: GUI of the tool.

Part III

Evaluation and Perspectives

Chapter 9

Evaluation

In this chapter we describe the evaluation approaches we considered in order to validate our procedure. Our first approach is explained in Section 9.1 while our second approach is explained in Section 9.2. Finally in Section 9.3 we describe the threats to the validity of our study and try to mitigate them.

9.1 Golden Answer Set

Our first approach to evaluate the resulting FMs was to try to find a *golden answer set* to compare the FM against. We have asked two groups of users, all of whom had a background in computer science, to manually construct an FM for the *antivirus* domain. Both groups were given a brief introduction to feature modeling so that they knew what was expected of them. This introduction included knowledge about the hierarchy, optional and mandatory features as well as OR-groups. We did not include exclusion clauses nor CTCs as the former could not be mined by our procedure and the latter would have been too time-consuming and was not of much use as what interested us most was to compare the hierarchy of the resulting FMs.
To perform the task, each group was given the same set of features extracted by the feature mining algorithm and was asked, during different sessions, to build an FM on a whiteboard. Both groups took approximately four hours to complete the task.

In both cases, the users started by organizing the features on a table to look at all of them individually and get a better understanding of the domain. As these features were numerous, both groups then started grouping them according to common topics. As the names of the features had been automatically mined, some of them were confusing, which led to some small variations in the features not always being grouped with the same other features. Moreover, he hierarchy within those groups as well as between the groups was quite different. At that point we realized that there was no *golden answer*

set to compare our FM against. This brings us to our next evaluation method which we will explain in the following section.

9.2 User Evaluation

To cope with the *golden answer set* problem, we chose to evaluate our procedure with a direct approach which involved asking graduate students who were familiar with feature modeling to take part in a survey during which they had to evaluate parts of four different FMs. These four FMs are as follow:

1. One automatically generated FM using only association rules (*probabilistic* approach);
2. One automatically generated by our algorithm (*clustered* approach);
3. Two manually constructed FMs (see Section 9.1).

We then created four different surveys which each included questions about parts of the four FMs. The participants were not informed of the source of the question to avoid creating bias in their judgement. The questions that were asked were always about either a specific parent-child relationship or a grouping of automatically generated features such as the one depicted in Figure 9.1. The surveys contained 25 questions each and were each completed by two users in approximately one hour.

9.2.1 Questions asked

We created three types of questions to evaluate different qualities of the FMs. The first type of question was created for the purpose of evaluating the quality of groups of features and was formulated in the following way: *"On a scale of 1-5 with 5 being the HIGHEST, please provide an overall rating for the group as a whole. A score of 5 means that the group is cohesive (i.e., all features belong together in the group), while a score of 1 means that the group is not very meaningful at all".*

The second type of question was created for the purpose of evaluating the quality of the parents for each grouping of features and was formulated in the following way: *"On a scale of 1-5 with 5 being the HIGHEST, please provide an overall rating for the parent of the group. A score of 5 means that the parent captures the essence of the features in the group and a score of 1 means that there is no obvious relationship between the parent and its child nodes".*

The third type of question was created for the purpose of evaluating parent-child relationships. The users were asked to check whether the associations were correct or not. For every group of features, we listed all the associations (as depicted in Figure 9.2).

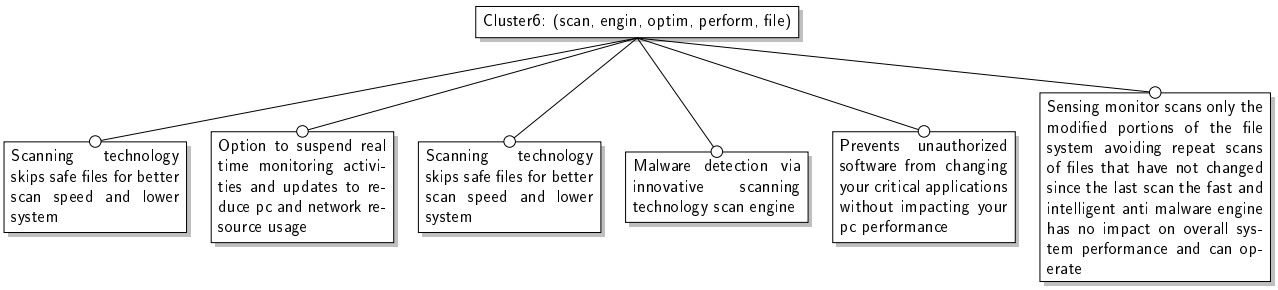


Figure 9.1: Group of features shown in the evaluation

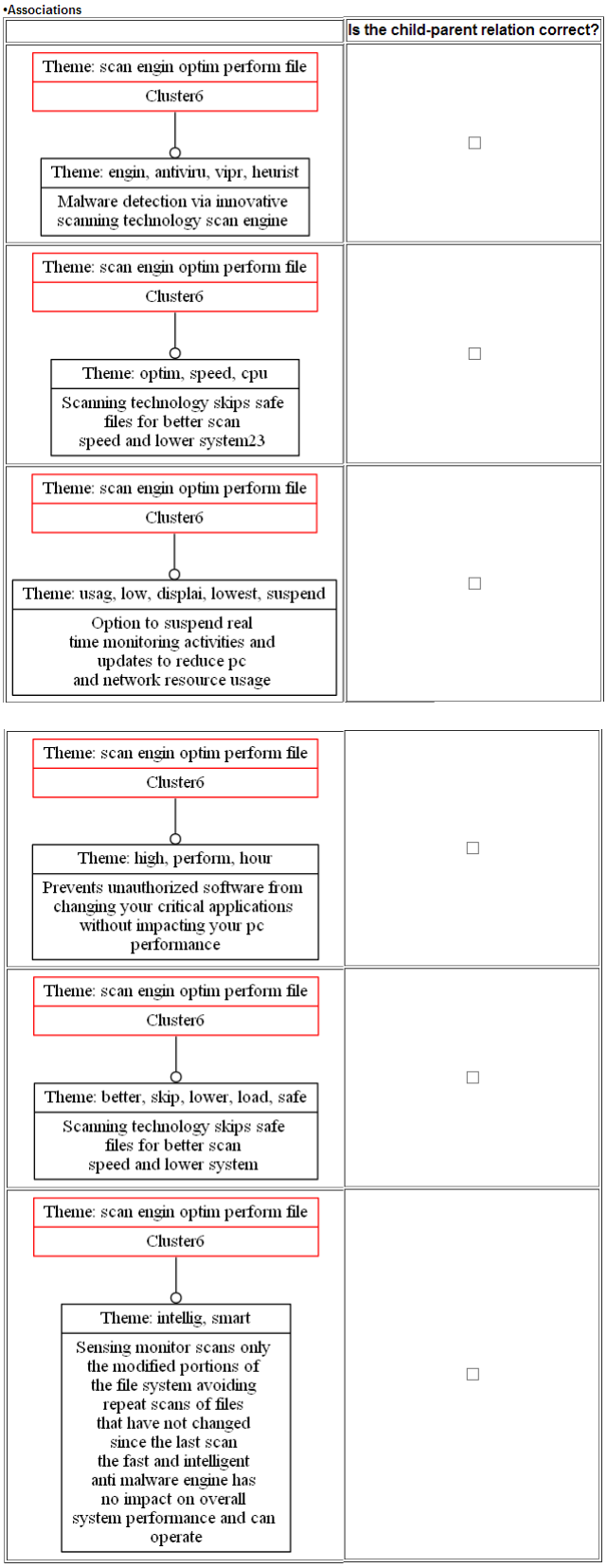


Figure 9.2: Parent-child relationships of the group of features in Figure 9.1

9.2.2 Results

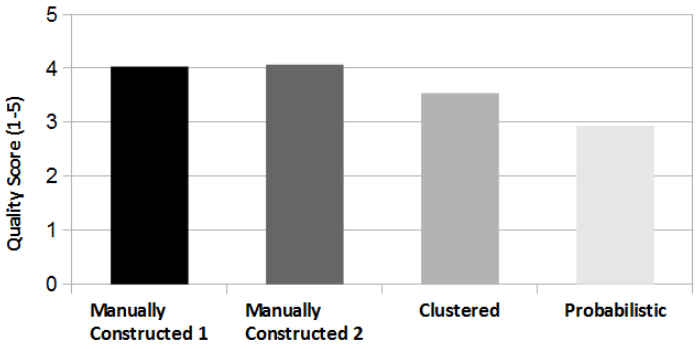
We evaluated three different qualities for the four different FMs (two manually constructed, one using the *clustered* approach and one using the *probabilistic* approach). These three qualities were evaluated using the three types of questions we designed and explained in the previous section. The results of those questions are reported respectively in Figures 9.3(a), 9.3(b) and 9.3(c).

Before we explain three figures reporting the results, we give a few metrics about the FMs.
Those FMs were all created using the same list of 80 features. The first group of users who created an FM decided to add 12 abstract features (see Section 7.3.2) to this list). The second group decided to add 6 abstract features. The clustered FM contained 7 abstract features, while the probabilistic FM contained none.

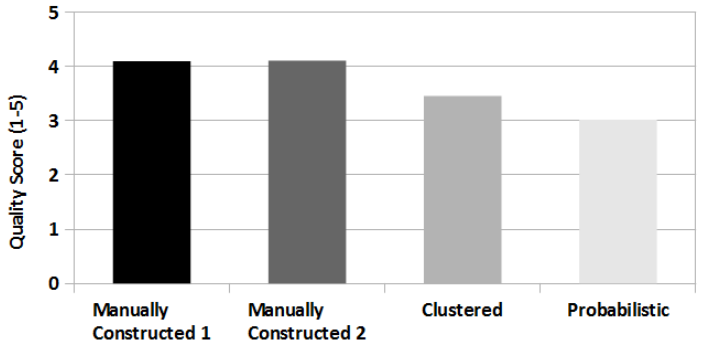
We did not ask the two groups of users to find CTCs in the FMs because, as we explained, that would have taken too much time. Indeed, there are 349 CTCs in the clustered FM and 329 in the probabilistic FM.
Those metrics as well as some others (number of leaf features, number of branches and depth of tree) are reported in Table 9.1.

Table 9.1: Metrics about the four FMs

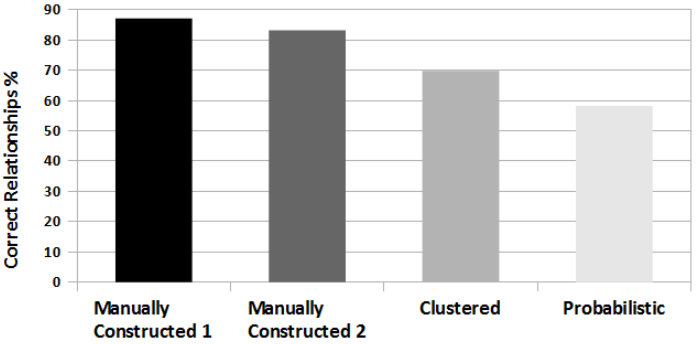
Metrics \ **FMs**	**Manually Constructed 1**	**Manually Constructed 2**	**Clustered**	**Probabilistic**
Number of Concrete Features	80	80	80	80
Number of Abstract Features	12	6	8	0
Number of Leaf Features	62	65	62	63
Number of Branches	18	19	4	8
Number of CTCs	0	0	349	329
Depth of Tree	7	5	6	4



(a) User evaluation of the quality of groups of features



(b) User evaluation of the quality of the parent for each group of features



(c) User evaluation of the quality of the individual parent-child relationships

Figure 9.3: A comparison of our approach: clustered *versus* manually created *versus* probabilistic approach for FM construction

Figure 9.3(a) shows that the people who participated in the survey rated the first manually constructed FM 4.03 (on a scale of 1-5 with 5 being the highest), 4.07 for the second manually constructed FM, 3.54 for the clustered FM and only 2.94 for the probabilistic FM regarding the quality of the groups.

Figure 9.3(b) shows that the people who participated in the survey rated the first manually constructed FM 4.10 (on a scale of 1-5 with 5 being the highest), 4.11 for the second manually constructed FM, 3.46 for the clustered FM and only 3.02 for the probabilistic FM regarding the quality of the parents of each group.

Figure 9.3(c) shows that the people who participated in the survey deemed that there were 87% of correct associations in the first manually constructed FMs, 83 for the second manually constructed FM, 70% in the clustered FM and only 58% in the probabilistic FM.

By examining those results we can clearly see that while the quality of feature groupings in the FM automatically generated by our approach (*clustered* FM) does not reach the one of the manually constructed FMs, it still outranks the feature groupings of the probabilistic FM. The same observation can be made for the two other qualities that we have evaluated, namely the quality of the parent for each group of features and the quality of the individual parent-child relationships.

Though we do not reach the same level of quality as the one obtained by manually constructing FMs, we can still say that our approach is useful because of the time-consuming and error-prone nature of the manual task. During our study, we asked two groups of users to manually construct an FM given a list of features. They did not have to think about CTCs, which would have taken even more time, and yet the task of grouping the features and finding a reasonably good hierarchy took each group approximately four hours to complete. Our algorithm is thus useful to quickly and reasonably accurately gain some insight about the domain.

9.3 Threats to Validity

Our evaluation study, just like any other evaluation, is exposed to threats. We will present the main threats in this section.

9.3.1 External Threat

The first and most important threat comes from the fact that we assumed that all the descriptions and set of features on which we based our whole process was a fair and representative view of the domain. The reason that led us to believe this was that the software download platform we used (Softpedia) is one of the most popular platform that millions of people use to download the software products they need. The fact that people trust and use this platform is also the reason why so many companies choose to put their software products on it to reach a larger number of people. As of April 2013, there were 1.4+ millions of items listed on their website and 2.7+ billions of downloads. Therefore we assumed that this was representative enough of the domain.

The domain we chose for the evaluation is *Antivirus*. Softpedia lists 165 different antivirus products for a total number of *feature descriptors* of 4,396. We also have to clarify that our method, but also most machine learning techniques are constrained by the fact that we can only learn from available data.

9.3.2 Construct Validity

The second threat to validity of our survey is its scope and is due to the time-consuming nature of the task that users need to perform. As we mentioned earlier, two groups of users had to manually create an FM so that we could ask questions to other persons regarding the various qualities of the FMs that were automatically generated versus the ones that were manually built. It is also worth mentioning that completing the user study takes approximately one hour. A larger study would be needed to evaluate various domains with more human users.

9.3.3 Internal Threats

The third threat is the computation of SCCs to identify AND-groups for a set of partially complete product descriptions. As explained in Section 6, Andersen et al. [ACSW12], compute SCCs in IGs to detect AND-groups. The mapping from SCCs to AND-groups is motivated by the fact that the implications between the features in the IG are transitive. Because there is a path between any two features in an SCC, each presence of a feature of an SCC in a configuration implies the presence of every other feature of the SCC in this configuration. However, the IGs in our case are not made of implications but probabilistic

association rules which are not transitive. It follows that features that do not occur often together in configurations may be considered as parts of the same AND-group. So, in case of an implication graph using probabilistic association rules, AND-groups can be computed from SCCs to offer the user an approximation of potential AND-groups but these groups must be manually reviewed.

The fourth and final threat is the evaluation process itself. Our procedure mines feature names automatically and those names might be either misleading or confusing. When we asked users to evaluate the quality of the associations within the FMs, it was impossible for us to separate the task of evaluating the quality of a feature's name and the actual quality of the association. Let us say we ask one of the participants whether the feature P is a correct parent of the child feature C. His choice might then be influenced by the real sense of the feature but also by the actual name of the feature. Even tough this can be a problem, it was essential to have a name for the features for human cognition purposes. To be able to mitigate the problem, we included both the automatically extracted feature name and the most occurring words that represent the theme of the underlying *feature descriptors* which resulted in this feature.

Chapter 10

Conclusion

In this chapter, we present a brief summary of our work. Then we present a critical outlook and finally we discuss about future research work that could be done based on our findings.

10.1 Summary

We have first shown the importance and the time-consuming nature of the domain analysis phase in software engineering. This phase is of crucial importance because it lays the foundations of the future software products that will arise from the development of the SPL. In Chapter 3 we have seen several existing techniques that extract FMs from either formally defined datasets or textual descriptions. However, those techniques could not be applied to our dataset because of its incomplete nature. We have then explained our novel approach in details in Chapters 4 to 7. This novel approach uses text mining, data mining and clustering algorithms to extract FMs from informal product descriptions that can be found on public software repositories such as *Softpedia*, *Cnet* or *MajorGeeks*. Finally we have presented an evaluation of our extraction process, in which we have shown that applying it on the antivirus domain led to clear improvements of the extracted FM compared to the probabilistic approach.

10.2 Critical Outlook

- During the development of the FM extraction procedure, we faced the challenge of automatically selecting a hierarchy for an FD among a large number of features. We took advantage of the textual descriptions associated to the features and incorporated text-mining techniques to ensure that the hierarchy provided

meaningful information regarding the targeted domain and to increase its maintainability. It would have been interesting to experiment further ideas based on text-mining techniques to enhance the quality of the extracted models. It would be especially interesting to evaluate the use of grammatical pattern-based or lexicon-based approaches to detect information related to variability among products such as techniques developed in [WCR09] or [HC06] and to compare the results to those obtained with our procedure.

- Further time should have been invested in the investigation of the utility of the extracted models. A qualitative evaluation could have provided relevant insight (see Section 10.3). Studying the utility of FMs for performing domain engineering tasks is time-demanding as it requires the manual construction, analysis or use of FMs that, in our context, contain a lot of features.
- During the design of the procedure, we were not concerned with the numbers of configurations in the dataset that were valid regarding the extracted FMs. Neither did we pay attention to the maximality property ([CSW08], see Section 7.5). While we were focusing on the presentation of meaningful information to domain analysts, it would have been interesting to consider these properties in our context.
- We focused on designing an automated procedure that can compute a hierarchy for FDs. The idea is to provide the user with an approximation of an FM that can be manually refined afterwards. It would be interesting to study how the addition of interactive mechanisms in the procedure impacts the quality of the extracted FMs.

10.3 Future Work

In this section, we address some of the weaknesses of our work regarding the extraction procedure that could be addressed in future work.

10.3.1 Further Evaluation

The principal further work that needs to be done is to apply and evaluate our approach on more domains, with a larger group of evaluators.

Comparing our approach to the other techniques presented in Chapter 3 could also be included in this further evaluation, but as we have mentioned when addressing the limits of the related work, these techniques have made the assumption that the products are formally described as sets of features and have not been specifically designed for large collections of incomplete textual descriptions. Applying them to the dataset we considered in our work would require some modifications.

As the main goal of the research was to study the extraction of FMs in order to support domain engineering tasks, a qualitative evaluation would be helpful to judge the impact of using our procedure to facilitate these tasks. It would be especially interesting to evaluate how the extracted raw FMs could be used as a starting point to build more refined FMs.

10.3.2 Human intervention

So far, we have relied only on available documentations to keep the FM extraction process described in Part II fully automated. It would be interesting to involve human intervention to allow users to adjust the structure of the FM hierarchy and to review the variability points. While the process would still be able to produce a complete FM entirely on its own, users could enhance the result by using their personal knowledge of the domain at various points in the procedure. For instance, at the end of the features clustering phase (see Section 7.2), the users could be allowed to reorganize the clusters of features by moving the features from one cluster to another. They could also review and edit AND-groups and OR-groups by removing or adding features or by choosing parents for the groups. They could change the features hierarchy by editing the final FM. They could also add new information that had not been discovered by the automated process such as XOR-groups or exclude-edges and they could even add new concrete and abstract features.

Manually extracting FMs from textual documentation is an arduous and error-prone task. By combining human intervention with the FM extraction procedure, a domain analyst would have automated support to help in this task and could enhance the quality and the completeness of the resulting FMs by involving his own knowledge of the domain in the process.

10.3.3 Feature Name Selection

Name selection for mined abstract features still needs to be addressed. In Section 7.3.2, we explain how the terms that received the highest tf-idf weight are used to form the name of the abstract feature. It would be more appropriate to say that these mined abstract features receive a label rather than a proper name as it simply consists of the conjunction of the terms. We believe that a proper name selection for the abstract features would enhance the readability of the mined FMs.

We could also use human intervention to enhance the quality of the extracted FMs by renaming features. The names of the features are automatically chosen in the features mining phase (see Section 5.4). While an automatically found feature name can accurately capture the aspects covered by feature in the product line, the user may want to edit the name so that it concisely and accurately represents a specific feature.

10.3.4 Configure products

In this thesis, we did not directly address the question of using the mined FMs to configure products. We can identify two relevant topics about product configuring for the FMs resulting from our procedure.

The first one is staged configuration. A staged configuration consists of a sequence of configuration choices. In [CHE04] and [CHE05], Czarnecki et al. introduce the concept of staged configuration which "*is achieved by the stepwise specialization of feature models or by multi-level configuration, where the configuration choices available in each stage are defined by separate feature models*". In [CHE04], Czarnecki et al. define the process of staged configuration "*as the removal of possible configuration choices*". The authors also define the process of *specialization* for an FD : "*Specialization is a transformation process that takes a feature diagram and yields another feature diagram, such that the set of the configurations denoted by the latter diagram is a true subset of the configurations denoted by the former diagram*". Staged configuration allows to configure products in multiple stages that can be executed at different moments by different people and where each stage eliminates some configuration choices. In [CHE04], Czarnecki et al. describe six different specialization steps to apply on FDs and one of them is called *unfolding a feature diagram reference*. Unfolding an FD reference consists in substituting a reference in an FD for the entire FD it refers to by means of its root feature. We believe that we could take advantage of our feature clustering approach to structure the FD (see Chapter 7) to apply a staged configuration based on unfolding FDs references. The first stage of the configuration would consist of making configuration choices on the FD made of the abstract features. Once the user has selected a subset of the abstract features, he can now make configuration decisions on the FD made of the selected abstract features and their descendant concrete features that have been unfolded below them. The abstract features would be used to make high-level configuration decisions in the first stage and the unfolded concrete features would be used to configure the product more precisely in the second stage. Figure 10.1 and Figure 10.2 show the two FDs for a staged configuration scenario.

The second topic about configuration which our mined FMs can address is *dynamic configuration*. *Dynamic configuration* from soft constraints in FMs has been first introduced in [CSW08] by Czarnecki et al.. Because our mined FMs are made of a lot of soft constraints (i.e., constraints with a confidence under 100%), it is possible to dynamically advise users to configure their product based on their previous configuration choices and the probabilities given by the soft constraints. For example, if users follow the recommendations made by a tool based on one FM resulting from our procedure and if the feature *mail attachment analysis* is only present in 15% of the product configurations, this feature will not be initially recommended to the users. However, if they have already selected the feature *antispam* and if a soft constraint indicates that when a product configuration has the feature *antispam*, it also has the feature *mail attachment analysis* 95% of the time (*mail attachment analysis* given *antispam* [95%]) then the tool will recommend the

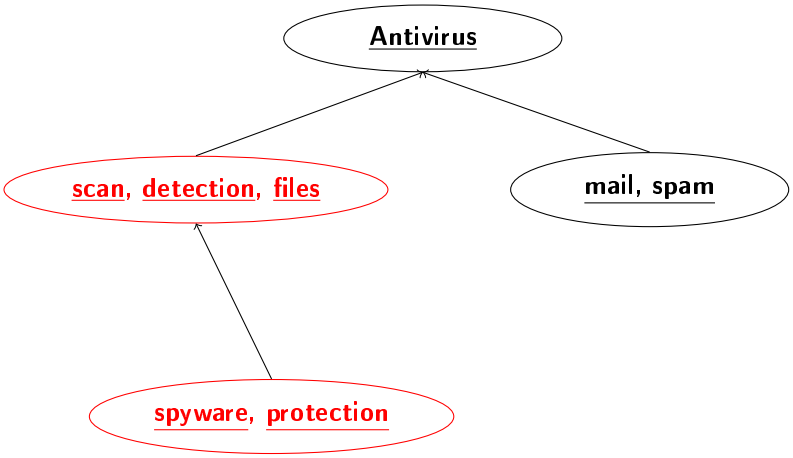


Figure 10.1: At the first configuration stage, the features *spyware, protection* and *scan, detection* are selected.

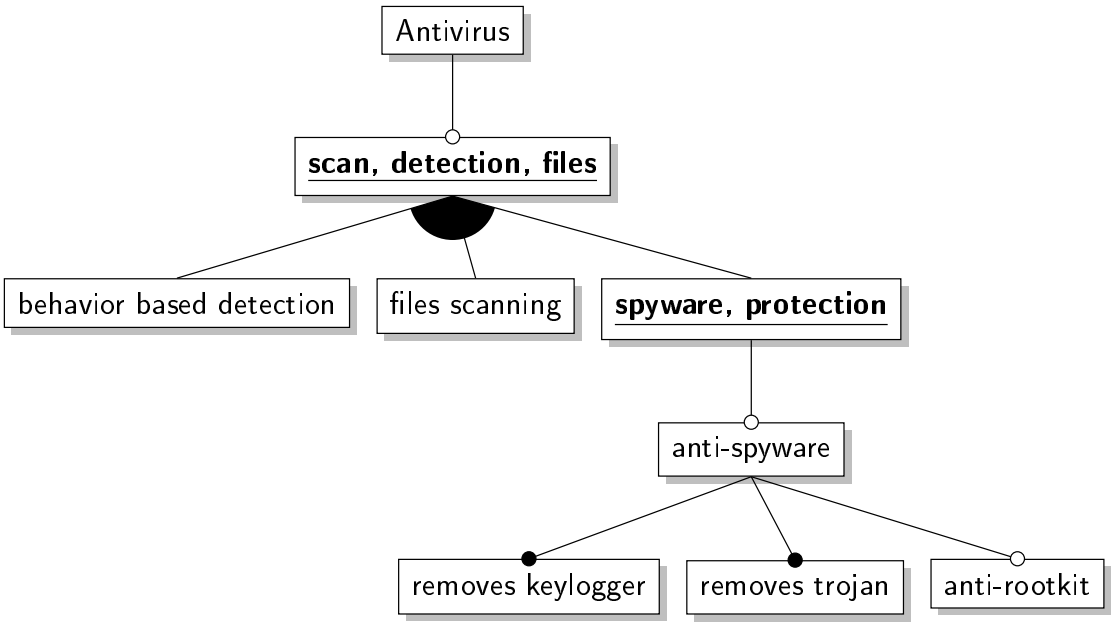


Figure 10.2: The FD that will be used in the second configuration stage. It is made of the abstract nodes that were selected during the first stage and their descendant concrete features.

feature *mail attachment analysis*. Dumitru et al. [DGH+11] study the development of a feature recommender system based on a product-by-feature matrix.

Bibliography

[AAP00] Ramesh C. Agarwal, Charu C. Aggarwal, and V. V. V. Prasad. A tree projection algorithm for generation of frequent itemsets. *Journal of Parallel and Distributed Computing*, 61:350–371, 2000.

[AB11] Sven Apel and Dirk Beyer. Feature cohesion in software product lines: an exploratory study. In *Proceedings of ICSE'11*, pages 421–430, New York, NY, USA, 2011. ACM.

[ACLF13] Mathieu Acher, Philippe Collet, Philippe Lahire, and Robert France. Familiar: A domain-specific language for large scale management of feature models. *Science of Computer Programming (SCP) Special issue on programming languages*, page 22, 2013.

[ACP+12] Mathieu Acher, Anthony Cleve, Gilles Perrouin, Patrick Heymans, Charles Vanbeneden, Philippe Collet, and Philippe Lahire. On extracting feature models from product descriptions. *Proceedings of the Sixth International Workshop on Variability Modeling of Software-Intensive Systems - VaMoS '12*, pages 45–54, 2012.

[ACSW12] Nele Andersen, Krzysztof Czarnecki, Steven She, and Andrzej Wąsowski. Efficient synthesis of feature models. *Proceedings of the 16th International Software Product Line Conference on - SPLC '12 -volume 1*, page 106, 2012.

[AIS93] Rakesh Agrawal, Tomasz ImieliÅĎski, and Arun Swami. Mining association rules between sets of items in large databases. *Proceedings of the 1993 ACM SIGMOD international conference on Management of data - SIGMOD '93*, (May):207–216, 1993.

[AK09] S. Apel and C. Kästner. An overview of feature-oriented software development. *Journal of Object Technology (JOT)*, 8(5):49–84, July/August 2009.

[AS94] R. Agrawal and R. Srikant. Fast algorithms for mining association rules. In *Proc. of the 20th Intn'l Conf. on Very Large Data Bases (VLDB'94)*, pages 487–499, Santiago, Chile, September 1994.

[BG11] E Bagheri and D Gasevic. Assessing the maintainability of software product line feature models using structural metrics. *Software Quality Journal*, 39:1–39, 2011.

[Big] BigLever – Gears. http://www.biglever.com/solution/product.html.

[BRN+13] Thorsten Berger, Ralf Rublack, Divya Nair, Joanne M. Atlee, Martin Becker, Krzysztof Czarnecki, and Andrzej Wąsowski. A survey of variability modeling in industrial practice. In *Proceedings of VaMoS'13*. ACM, 2013.

[BSRC10] David Benavides, Sergio Segura, and Antonio Ruiz-Cortés. Automated analysis of feature models 20 years later: A literature review. *Information Systems*, 35(6):615–636, September 2010.

[CBH11] Andreas Classen, Quentin Boucher, and Patrick Heymans. A text-based approach to feature modelling: Syntax and semantics of tvl. *Sci. Comput. Program.*, 76(12):1130–1143, 2011.

[CE00] Krzysztof Czarnecki and Ulrich W. Eisenecker. *Generative programming: methods, tools, and applications*. ACM Press/Addison-Wesley Publishing Co., New York, NY, USA, 2000.

[CGR+12] Krzysztof Czarnecki, Paul Grünbacher, Rick Rabiser, Klaus Schmid, and Andrzej Wąsowski. Cool features and tough decisions: a comparison of variability modeling approaches. In *Proceedings of VaMoS'12*, pages 173–182, New York, NY, USA, 2012. ACM.

[CHE04] Krzysztof Czarnecki, Simon Helsen, and Ulrich Eisenecker. Staged configuration using feature models. *Software Product Lines*, 2004.

[CHE05] Krzysztof Czarnecki, Simon Helsen, and Ulrich Eisenecker. Staged configuration through specialization and multi-level configuration of feature models. In *Software Process Improvement and Practice*, page 2005, 2005.

[CHS08a] Andreas Classen, Patrick Heymans, and Pierre-Yves Schobbens. What's in a feature: A requirements engineering perspective. In *Proceedings of FASE'08*, volume 4961 of *LNCS*, pages 16–30, 2008.

[CHS08b] Andreas Classen, Patrick Heymans, and Pierre-Yves Schobbens. What's in a feature: a requirements engineering perspective. In *Proceedings of the Theory and practice of software, 11th international conference on Fundamental approaches to software engineering*, FASE'08/ETAPS'08, pages 16–30, Berlin, Heidelberg, 2008. Springer-Verlag.

[CN01] Paul C. Clements and Linda Northrop. *Software Product Lines: Practices and Patterns*. SEI Series in Software Engineering. Addison-Wesley, 2001.

[CSW08] Krzysztof Czarnecki, Steven She, and Andrzej Wąsowski. Sample Spaces and Feature Models: There and Back Again. *2008 12th International Software Product Line Conference*, pages 22–31, September 2008.

[CW07] Krzysztof Czarnecki and Andrzej Wąsowski. Feature Diagrams and Logics: There and Back Again. *11th International Software Product Line Conference (SPLC 2007)*, pages 23–34, September 2007.

[CZZM05] Kun Chen, Wei Zhang, Haiyan Zhao, and Hong Mei. An approach to constructing feature models based on requirements clustering. Technical report, 2005.

[DGH+11] Horatiu Dumitru, Marek Gibiec, Negar Hariri, Jane Cleland-Huang, Bamshad Mobasher, Carlos Castro-Herrera, and Mehdi Mirakhorli. On-demand feature recommendations derived from mining public product descriptions. *Proceeding of the 33rd international conference on Software engineering - ICSE '11*, page 181, 2011.

[DM01] Inderjit S. Dhillon and Dharmendra S. Modha. Concept decompositions for large sparse text data using clustering. *Journal of Machine Learning*, 42:143–175, January 2001.

[FPDF98] William Frakes, Ruben Prieto-;Diaz, and Christopher Fox. Dare: Domain analysis and reuse environment. *Annals of Software Engineering*, 5(1):125–141, 1998.

[GCI79] M. R. Garey, D. S. Johnson. Computers, and Intractability. M. Computers and Intractability: A Guide to the Theory of NP-Completeness. M. 1979.

[Han05] Jiawei Han. *Data Mining: Concepts and Techniques*. Morgan Kaufmann Publishers Inc., San Francisco, CA, USA, 2005.

[HC06] Ya-Han Hu and Yen-Liang Chen. Mining association rules with multiple minimum supports: a new mining algorithm and a support tuning mechanism. *Decision Support Systems*, 42(1):1–24, October 2006.

[HL04] Minqing Hu and Bing Liu. Mining and summarizing customer reviews. In *Proceedings of KDD'04*, pages 168–177, New York, NY, USA, 2004. ACM.

[HPY00] J. Han, J. Pei, and Y. Yin. Mining frequent patterns without candidate generation. In *Proceedings of SIGMOD'00*, New York, NY, June 2000.

[IFFD10] Syed Imran, Franclin Foping, John Feehan, and IM Dokas. Domain Specific Modeling Language for Early Warning System: Using IDEF0 for Domain Analysis. *IJCSI International Journal of ...*, 7(5):10–17, 2010.

[Ins] Software Engineering Institute. Software product lines, http://www.sei.cmu.edu/productlines/.

[J.76] MacQUEEN J. Some methods for classification and analysis of multivariate observations. *Proceedings of 5-th Berkeley Symposium on Mathematical Statistics and Probability*, pages 281–297, 1976.

[KCH+90] K. C. Kang, S. G. Cohen, J. A. Hess, W. E. Novak, and A. S. Peterson. Feature-oriented domain analysis (foda) feasibility study. Technical report, Carnegie-Mellon University Software Engineering Institute, November 1990.

[LHM99] Bing Liu, Wynne Hsu, and Yiming Ma. Mining association rules with multiple minimum supports. *Proceedings of the fifth ACM SIGKDD international conference on Knowledge discovery and data mining - KDD '99*, pages 337–341, 1999.

[Nei80] James Milne Neighbors. *Software construction using components*. PhD thesis, 1980. AAI8106784.

[oST93] National Institute of Standards and Technology. Integration definition for information modeling (idef0). 1993.

[PBvdL05] Klaus Pohl, Günter Böckle, and Frank van der Linden. *Software Product Line Engineering: Foundations, Principles and Techniques*. Springer-Verlag New York, Inc., Secaucus, NJ, USA, 2005.

[Por80] Porter. *Program 14*, pages 130–137, 1980.

[pur] pure::variants. http://www.pure-systems.com/pure_variants.49.0.html.

[SCRR05] Américo Sampaio, Ruzanna Chitchyan, Awais Rashid, and Paul Rayson. Ea-miner: a tool for automating aspect-oriented requirements identification. In *Proceedings of the 20th IEEE/ACM international Conference on Automated software engineering*, ASE '05, pages 352–355, New York, NY, USA, 2005. ACM.

[Sil97] J. P. M. Silva. On computing minimum size prime implicants. 1997.

[SLB+11] Steven She, Rafael Lotufo, Thorsten Berger, Andrzej Wąsowski, and Krzysztof Czarnecki. Reverse engineering feature models. *Proceeding of the 33rd international conference on Software engineering - ICSE '11*, page 461, 2011.

[SV02] Klaus Schmid and Martin Verlage. The economic impact of product line adoption and evolution. *IEEE Software*, 19(4):50–57, 2002.

[SvGB05] Mikael Svahnberg, Jilles van Gurp, and Jan Bosch. A taxonomy of variability realization techniques. *Softw., Pract. Exper.*, 35(8):705–754, 2005.

[Tar77] R. E. Tarjan. Finding optimum branchings. *Networks*, 7(1):25–35, 1977.

[TKB+12] Thomas Thüm, Christian Kästner, Fabian Benduhn, Jens Meinicke, Gunter Saake, and Thomas Leich. Featureide: An extensible framework for feature-oriented software development. *Science of Computer Programming (SCP)*, 2012.

[VMMO] J. P. M. Silva V. M. Manquinho, P. F. Flores and A. L. Oliveira. Prime implicant computation using satisfiability algorithms. *ICTAI. IEEE Computer*.

[WCR09] Nathan Weston, Ruzanna Chitchyan, and Awais Rashid. A framework for constructing semantically composable feature models from natural language requirements. *Proceedings of the 13th International . . .*, pages 211–220, 2009.

[WL99] D.M. Weiss and C.T.R. Lai. *Software product-line engineering: a family-based software development process*. Addison-Wesley, 1999.

[Zak00] Mohammed J. Zaki. Scalable algorithms for association mining. *IEEE Transactions on Knowledge and Data Engineering*, 12(3):372–390, 2000.

Part IV

Appendices

Appendix A

Raw feature descriptors clusters and feature names

A.1 Clustered feature descriptors

Listing A.1: This is a portion of five clusters of *feature descriptors* that have been put together by the SPKMeans algorithm.

```
1 cluster - theme: 'updat, databas, automat, viru, definit'
ID      Text
2104    Virus definition update and automatic update supported
956     Automatically updates the software and virus database
1104    Automatic database updates
1145    Automatic database updates
1173    Automatic database updates
651     Automatic incremental updates for virus database and program
    modules
936     Automatic incremental database updates
702     Automatic updating
1860    Automatic updates
2179    Incremental virus database update
2192    Incremental virus database update
2030    Automatic Update via internet
1329    Automatic updates over the Internet complete product and
    incremental updates of definition files
1027    Automatic updation of program and definition files
2187    Daily virus database updates
2198    Daily virus database updates
1861    Schedule automatic upgrades and automatic updates of virus
    signatures and definitions AntiVirus for MDaemon automatically
    downloads and installs updates for hands free administration
1944    Automatic ASAV signature updates
1325    Manual Update of Database
```

```
875       Virus signatures software updates and upgrades are
212       Updates
280       Updates
624       Automatic downloads of regularly updated Virus Database
470       Virus Defintions Updated Manual Daily
1614      Virus Database updates Virus database updates which are compact in
     size are released everyday Added to this emergency updates are
    released whenever there is an outbreak of a virus in the wild
    InstaUpdate feature of Protector Plus 2007 will automatically download
    these updates free of cost
791       Automatic Compressed Updates are being provided by eScan for both
    the software as well as the virus and spam definitions The Virus
    signatures or spam definitions are being automatically updated hourly
    for instant protection from existing and emerging threats Multilanguage
     Support
488       Manual Definition Updates
343       Automatic updates are another key point in virus protection both
    the virus database and the program itself can be updated automatically
    The updates are incremental and only new data is downloaded thus
    reducing the transfer time The typical size of a virus database update
    is approximately of 20 80kb the program update usually has
    approximately 200 500kb
489       Manual Program Updates
...

2 cluster - theme: 'real, time, scan, demand'
ID        Text
164       Scans all web e mail and instant messaging traffic in real time
496       Scans all Web e mail and instant messaging traffic in real time
520       Scans all Web e mail and instant messaging traffic in real time
541       Scans all Web e mail and instant messaging traffic in real time
1067      Scans files in real time on access and on demand
1108      Scans files in real time on access and on demand
1149      Scans files in real time on access and on demand
1451      Scans files in real time and on demand
471       Real time scanning
1855      Real time File Monitoring
653       Real Time and On Demand scanning
95        Real time Scan detects and blocks security threats in real time to
     protect your system
174       Real time anti rootkit protection
238       Real time anti rootkit protection
2134      Real time process monitor
1418      IntelliGuard protects your computer against threats in real time
388       Resident virus guard real time scan or on access scan for constant
     monitoring of all file accesses
1610      Real time Scan The Real time scanner of Protector Plus 2007 is
    designed to monitor the system all the time It prevents any virus from
    entering the system during the activities like browsing the internet
    accessing the network or while using removable media like cd rom usb
    drives etc The Real time scanner will ensure that the system is free of
     any security threats
111       Monitoring files and data downloading from IE in real time
```

```
230       Real time graphic scanning reports
298       Real time graphic scanning reports
425       Real time monitoring of every file access through integrated on
    access scanner incl archive scanning as well as on demand scanner for
    manual and time driven search runs
1010      Real time scanner All new files accessed downloaded created or
    modified are automatically scanned Ensures complete protection at all
    times
1777      NEW Real Time Check of Internet Traffic
119       Real time malware firewall
201       Real time scanning of opened executed files
267       Real time scanning of opened executed files
2052      On demand schedules and real time scanning options actively
    protect your computer from viruses and spyware in files incoming and
    outgoing mail webmail and instant messages Customizable Security
    Warnings
369       LinkScanner blocks poisoned web pages in real time
...

3 cluster - theme: 'shield'
ID        Text
204       Web Shield
270       Web Shield
210       Network Shield
276       Network Shield
202       Mail Shield
268       Mail Shield
196       Behavior Shield
262       Behavior Shield
195       Shields
206       P2P Shield
261       Shields
272       P2P Shield
200       File System Shield
266       File System Shield
278       Script Shield
306       Script shield
208       IM Shield
274       IM Shield
861       OutbreakShield Immediate protection against new
964       OutbreakShield Immediate protection against new viruses
1463      Shields your PC from hackers on the Web
368       Web Shield screens downloads and IM for infections
315       Emailing chatting and downloading Web Shield
1885      IE Home Page Shield
1314      Because NovaShield uses policies rather than signatures to detect
    malicious behavior it does not require frequent updates
312       Ensures every web page you visit is safe even before you go there
    LinkScanner Search Shield
1312      NovaShield is lightweight in its use of system resources and
    therefore has minimal performance impact on your computer
...
```

```
4 cluster - theme: 'instant, messeng, messag, msn, yahoo, encrypt, convers,
    mail, traffic'
ID      Text
502     Instant Messaging Encryption keeps your conversations private on
    Yahoo and MSN Messenger
525     Instant Messaging Encryption keeps your conversations private on
    Yahoo and MSN Messenger
545     Prevents personal information from leaking via e mail Web or
    instant messagingNEW Guard your conversations with top of the line
    encryption
524     Prevents personal information from leaking via e mail Web or
    instant messagingNEW Guard your files and conversations with top of the
     line encryption
546     Instant Messaging Encryption keeps your conversations private on
    Yahoo and MSN Messenger Play safely play seamlessly
26      Guard your conversations with top of the line encryption for both
    Yahoo and MSN Messenger Game player options
55      Guard your conversations with top of the line encryption for both
    Yahoo and MSN Messenger Game player options
164     Scans all web e mail and instant messaging traffic in real time
496     Scans all Web e mail and instant messaging traffic in real time
520     Scans all Web e mail and instant messaging traffic in real time
541     Scans all Web e mail and instant messaging traffic in real time
1454    Scans Instant Messaging traffic in MSN Messenger Windows Live
    Messenger Yahoo Messenger and AOL
20      Guard your conversations with top of the line encryption for both
    Yahoo and MSN Messenger Optimized scanning technology
16      Acronis encrypts Yahoo and MSN Messenger instant messaging traffic
     to keep conversations private If you are a gamer Acronis AntiVirus
    2010 s user friendly interface can be set to eliminate interruptions
    during game play Scans faster to lower system overhead
46      Acronis encrypts Yahoo and MSN Messenger instant messaging traffic
     to keep conversations private If you are a gamer Acronis Internet
    Security 2010 s user friendly interface can be set to eliminate
    interruptions during game play Scans faster to lower system overhead
869     Virus blocker for instant messaging MSN Yahoo AOL etc
1070    Protects instant messengers ICQ MSN
1111    Protects instant messengers ICQ MSN
1152    Protects instant messengers ICQ MSN
165     Prevents leaking of personal data via e mail or instant messenger
500     Prevents personal information from leaking via e mail Web or
    instant messagingNEW
166     Encryption can keep your Yahoo or MSN Messenger chats private
854     IMAP instant messaging MSN Yahoo AOL etc
967     Virus blockers for e mails and instant messaging
96      Real time Messenger Scan blocks infected files and prevents
    installation of untrusted programs via instant messenger Security
    policies for network security
25      Communicate without fear of compromising your identity Instant
    Messaging Encryption
54      Communicate without fear of compromising your identity Instant
    Messaging Encryption
```

```
40      Eliminate interruptions during gaming and manage security for up
    to three PCs from a single management point Encrypts Yahoo and MSN
    Instant Messages
209     Checks files downloaded while using instant messaging or chat
    programs AIM AOL Instant Messenger Gadu Gadu gaim Pidgin Google Talk
    ICQ IM2 Messenger Infium Miranda mIRC MSN Windows Messenger Odigo
    PalTalk Messenger Psi Jabber Client QIP QQ SIM Skype Tlen Trillian
    WengoPhone XFire Yahoo Messenger
...

5 cluster - theme: 'spywar, adwar, anti, block'
ID      Text
1259    Spyware protection detects blocks and removes spyware and adware
2050    Powerful anti spyware technology guards your personal information
    and identity from spyware rootkits and other malicious software
    Prevents Unauthorized Changes
320     Prevents unauthorized information access by spyware and adware
    Anti Rootkit
2057    Powerful anti spyware technology guards your personal information
    and identity from spyware rootkits and other malicious software Anti
    Spam
2072    Powerful anti spyware technology guards your personal information
    and identity from spyware rootkits and other malicious software Anti
    Spam
1567    Spyware is malicious software that is installed on the system
    without the user s knowledge to gather personal information of the user
     and to monitor the critical information such as user name passwords
    bank account details credit card details etc The AntiSpyware feature
    blocks spywares before they get installed on your PC It also protects
    your privacy by detecting and cleaning spywares and blocking their
    activities of identity theft automatically AntiMalware
305     Anti spyware protection
1017    Anti Spyware Prevention of stealthy software installation
1598    Blocks spywares before they get installed on your PC Protects your
     privacy by detecting and cleaning spywares and blocking their
    activities of identity theft automatically Firewall
2049    Prevents unauthorized software from changing your critical
    applications without impacting your PC performance Anti Spyware
2056    Prevents unauthorized software from changing your critical
    applications without impacting your PC performance Anti Spyware
2071    Prevent unauthorized software from changing your critical
    applications without impacting your PC performance Anti Spyware
1579    Detects and cleans spywares adwares dialers and other malicious
    software and protects identity theft AntiRootkit
1066    Blocks spyware and adware
1107    Blocks spyware and adware
1148    Blocks spyware and adware
1247    Blocks Spyware Blocks spyware before it installs on your computer
    and removes existing spyware so you can surf the Web safely
987     3 Anti Spyware
1437    Protection against spyware Protects your privacy by preventing
    spyware from being installed on your computer The anti spyware
    protection prevents this type of software from compiling data about
```

```
    your Internet habits and preferences
1494        Protection against spyware Protects your privacy by preventing
    spyware from being installed on your computer The anti spyware
    protection prevents this type of software from compiling data about
    your Internet habits and preferences
619         Spyware Keep your data safe from spyware adware and identity theft
822         Anti Spyware Protect your privacy by preventing spyware and
    hackers from entering your computer
80          A Spyware Scan tracks down spyware programs previously installed
    on your computer and blocks new spyware from being installed Personal
    Firewall
481         Adware Spyware Scanning
2128        Prevents spyware from getting on your PC by blocking its primary
    source spyware distribution websites Privacy Protection
365         Antivirus and Anti Spyware protection against viruses worms
    spyware and trojans
1403        Comprehensive Spyware Detection and Removal Provides complete
    protection against spyware adware keyloggers Remote Access Trojans RATs
     and browser hijackers
637         Protects your PC from existing and new spyware threats Anti
    Phishing
885         Dangerous spyware can infect your system in many ways Prevent
    spyware from harming your computer with real time protection Scan
    automatically of on demand and remove existing spyware and malware Keep
     Your PC Running Better Faster amp Longer
...
```

Appendix B

Algorithms

B.1 Chu-Liu/Edmonds' Algorithm

Listing B.1: Chu-Liu/Edmonds' algorithm

```
package algorithm.edmonds;
import edu.uci.ics.jung.graph.util.EdgeType;
import edu.uci.ics.jung.graph.util.Pair;
import graph.Association;
import graph.ImplicationGraph;
import graph.nodes.Node;
import java.util.ArrayList;
import java.util.HashSet;
import java.util.Set;
import org.apache.logging.log4j.LogManager;
import org.apache.logging.log4j.Logger;
/**
 * Edmond's Maximum branching algorithm implementation based on <a href=
 * "http://cw.felk.cvut.cz/lib/exe/fetch.php/courses/a4m33pal/cviceni/
     tarjan-finding-optimum-branchings.pdf"
 * >Tarjan's paper</a>.
 *
 * @author Edouard Delfosse
 * @author Jean-Marc Davril
 */
public class EdmondTarjan {
  static Logger logger = LogManager.getLogger(EdmondTarjan.class.getName())
      ;
  private ImplicationGraph graph;
  /**
   * For each K in queueList, queue represent the edges incoming to this
       node
   * K
   */
```

```
private ArrayList<Queue> queueList = new ArrayList<Queue>();
private ArrayList<ConnectedComponent> stronglyConnectedComponents = new
    ArrayList<ConnectedComponent>();
private ArrayList<ConnectedComponent> weaklyConnectedComponents = new
    ArrayList<ConnectedComponent>();
private ArrayList<Node> roots = new ArrayList<Node>();
private ArrayList<Pair<Node>> h = new ArrayList<Pair<Node>>();
/**
 * Root components with no available incoming edges
 */
private ArrayList<Association> rset = new ArrayList<Association>();
/**
 * min(i) is the vertex which algorithm ROOT will select if applied to
     root
 * component i
 */
private ArrayList<Node> min = new ArrayList<Node>();
/**
 * @param graph
 */
public EdmondTarjan(ImplicationGraph graph) {
  this.graph = graph;
}
/**
 * Get the maximum branching graph from the list of pairs
 *
 * @return
 */
public ImplicationGraph getMaxGraph() {
  findMaximumBranching();
  return convertHToGraph("maxGraphInverted");
}
/**
 * Get the Maximum branching graph
 *
 * @return
 */
private ArrayList<Pair<Node>> findMaximumBranching() {
  initStep();
  generalStep();
  return h;
}
/**
 * Convert H to a graph
 *
 * @return
 */
private ImplicationGraph convertHToGraph(String graphName) {
  ImplicationGraph g = new ImplicationGraph();
  for (Pair<Node> pair : h) {
    if (!g.containsVertex(pair.getFirst()))
      g.addVertex(pair.getFirst());
    if (!g.containsVertex(pair.getSecond()))
```

```
        g.addVertex(pair.getSecond());
      g.addEdge(graph.findEdge(pair.getFirst(), pair.getSecond()),
          pair.getFirst(), pair.getSecond(), EdgeType.DIRECTED);
    }
    return g;
  }
  /**
   * Initialization step
   */
  private void initStep() {
    for (Node node : graph.getVertices()) {
      queueList.add(new Queue(new ArrayList<Association>(graph
          .getInEdges(node))));
      Set<Node> set = new HashSet<Node>();
      set.add(node);
      stronglyConnectedComponents.add(new ConnectedComponent(set, graph));
      weaklyConnectedComponents.add(new ConnectedComponent(set, graph));
      // enter(i) = (0,0)
      // this step is automatically done by setting inEdges to a new
      // ArrayList in the strongly connected component
      roots.add(node);
      min.add(node);
    }
  }
  /**
   * Algorithm
   */
  private void generalStep() {
    int k = 0;
    // The representation we chose for roots is an ArrayList, in which at
    // index k there is the node K.
    // When we take that node K out of the list, we set his position to
    // NULL.
    // But by doing that, the list is never empty, which means we have to
    // loop through it to check if there is still an element that needs to
    // be processed
    boolean isEmpty = false;
    // int index = 0;
    while (!isEmpty) {
      Node nodeK = roots.get(k);
      if (nodeK != null) {
        // We set the kth element of the list to null instead of
        // removing it to keep the list in the same order as our other
        // lists (corresponding indexes)
        roots.set(k, null);
        // (i,j) = MAX(k)
        Association assoc = queueList.get(k).max();
        // rset = rset U {k}
        if (assoc == null) {
          rset.add(assoc);
          continue;
        }
        // If the node (source of the association with the maximum
```

```
        // weight) was in the same strongly connected component as the
        // node K, we need to put it back in the list of roots
        // if SFIND(i) = k then roots = roots U {k}
        Node source = graph.getSource(assoc);
        Node dest = graph.getDest(assoc);
        if (k == findStronglyConnected(source)) {
          roots.set(k, nodeK);
          continue;
        }
        // H = H U {(i,j)}
        // logger.debug("Assoc = " + assoc + ", ruleNumber = "
        // + assoc.getRuleNumber() + ", number = "
        // + assoc.getNumber());
        // logger.debug("Source = " + source);
        // logger.debug("Dest = " + dest);
        // logger.debug("==========");
        h.add(new Pair<Node>(source, dest));
        /**
         * Index of the weakly connected component containing source
         */
        int weaklySource = findWeaklyConnected(source);
        /**
         * Index of the weakly connected component containing dest
         */
        int weaklyDest = findWeaklyConnected(dest);
        if (weaklySource != weaklyDest) {
          // Add all the nodes of the weakly connected component
          // containing source to the weakly connected component
          // containing source and then delete the former
          // WUNION(WFIND(i), WFIND(j))
          weaklyConnectedComponents.get(weaklySource).addToSet(
              weaklyConnectedComponents.get(weaklyDest).getCc());
          weaklyConnectedComponents.set(weaklyDest, null);
          // enter(k) = (i,j)
          stronglyConnectedComponents.get(k).setInEdge(
              new Pair<Node>(source, dest));

        } else {
          double val = Double.MAX_VALUE;
          // (x,y) = (i,j)
          Pair<Node> copy = new Pair<Node>(
              ImplicationGraph.cloneNode(source),
              ImplicationGraph.cloneNode(dest));
          // while (x,y) != (0,0)
          int vertex = -1;
          while (copy != null) {
            double confidence = graph.findEdge(source, dest)
                .getConfidence();
            if (confidence < val) {
              // val = c(x,y)
              val = confidence;
              // vertex = SFIND(y)
              vertex = findStronglyConnected(copy.getSecond());
```

```
            }
            copy = stronglyConnectedComponents.get(
                findStronglyConnected(copy.getFirst()))
                .getInEdge();
        }
        double confidenceIJ = graph.findEdge(source, dest)
            .getConfidence();
        // ADD(val-c(i,j),k)
        queueList.get(k).add(val / confidenceIJ);
        // min(k) = min(vertex)
        min.set(k, min.get(vertex));
        // (x,y) = enter(SFIND(i))
        copy = stronglyConnectedComponents.get(
            findStronglyConnected(source)).getInEdge();
        // while (x,y) != (0,0)
        while (copy != null) {
          // ADD(val-c(x,y),SFIND(y))
          double confidenceXY = graph.findEdge(copy.getFirst(),
              copy.getSecond()).getConfidence();
          queueList.get(findStronglyConnected(copy.getSecond()))
              .add(val / confidenceXY);
          // QUNION(k,SFIND(y))
          queueList
              .get(k)
              .getEdges()
              .addAll(queueList
                  .get(findStronglyConnected(copy
                      .getSecond())).getEdges());
          // SUNION(k,SFIND(y))
          int sFINDY = findStronglyConnected(copy.getSecond());
          stronglyConnectedComponents.get(k)
              .addToSet(
                  stronglyConnectedComponents.get(sFINDY)
                      .getCc());
          stronglyConnectedComponents.set(sFINDY, null);
          // (x,y) = enter(SFIND(x))
          copy = stronglyConnectedComponents.get(
              findStronglyConnected(copy.getFirst()))
              .getInEdge();
        }
        roots.set(k, nodeK);
      }

    }
    // convertHToGraph(Integer.toString(index));
    // index++;
    k = 0;
    // Check if there is still an element in the roots
    isEmpty = true;
    for (int j = 0; j < roots.size(); j++) {
      if (roots.get(j) != null) {
        isEmpty = false;
        break;
```

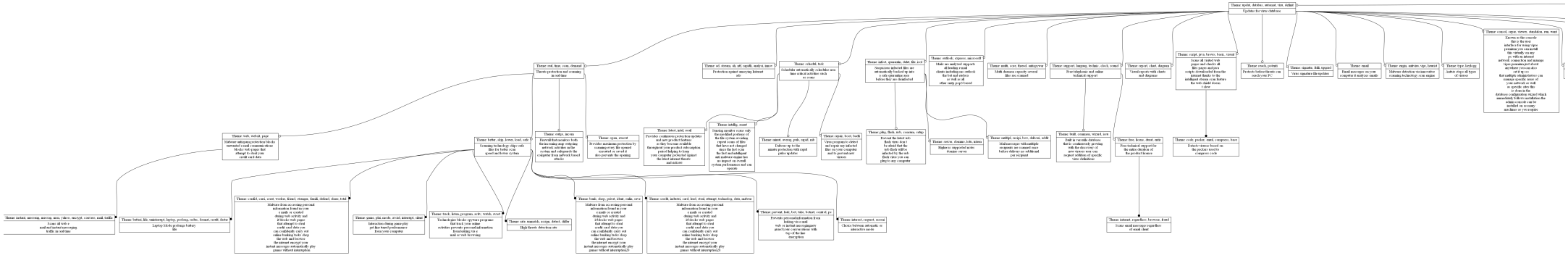
```
        }
        k++;
      }
    }
  }
  /**
   * Find the index of the strongly connected component containing the node
   * from
   *
   * @param from
   * @return index of the strongly connected component containing the node
   *         from
   */
  private int findStronglyConnected(Node from) {
    for (int i = 0; i < stronglyConnectedComponents.size(); i++) {
      if (stronglyConnectedComponents.get(i) != null) {
        if (stronglyConnectedComponents.get(i).getCc().contains(from)) {
          return i;
        }
      }
    }
    return -1;
  }
  /**
   * Find the index of the weakly connected component containing the node
      from
   *
   * @param from
   * @return index of the weakly connected component containing the node
      from
   */
  private int findWeaklyConnected(Node from) {
    for (int i = 0; i < weaklyConnectedComponents.size(); i++) {
      if (weaklyConnectedComponents.get(i) != null) {
        if (weaklyConnectedComponents.get(i).getCc().contains(from)) {
          return i;
        }
      }

    }
    return -1;
  }
}
```

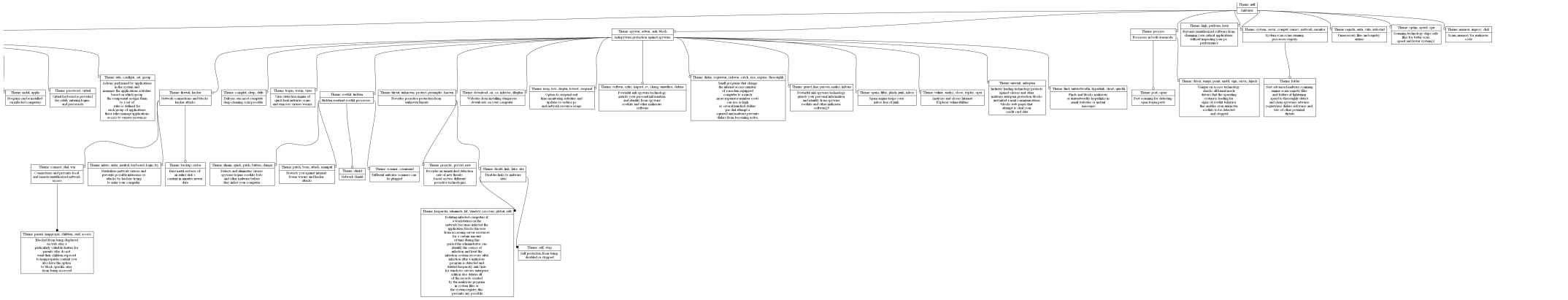
Appendix C

Feature Models

This chapter contains various FMs examples.

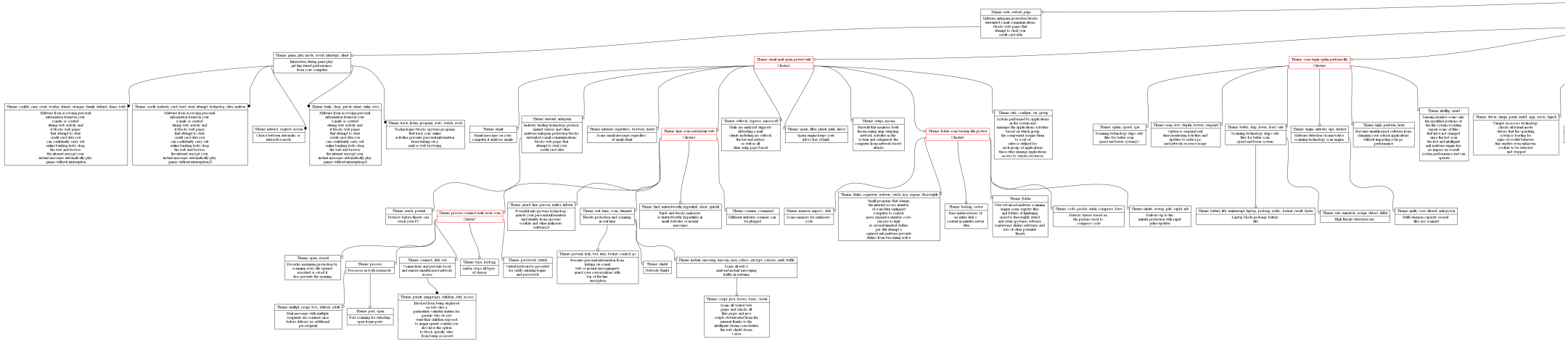


(a) (1/2)

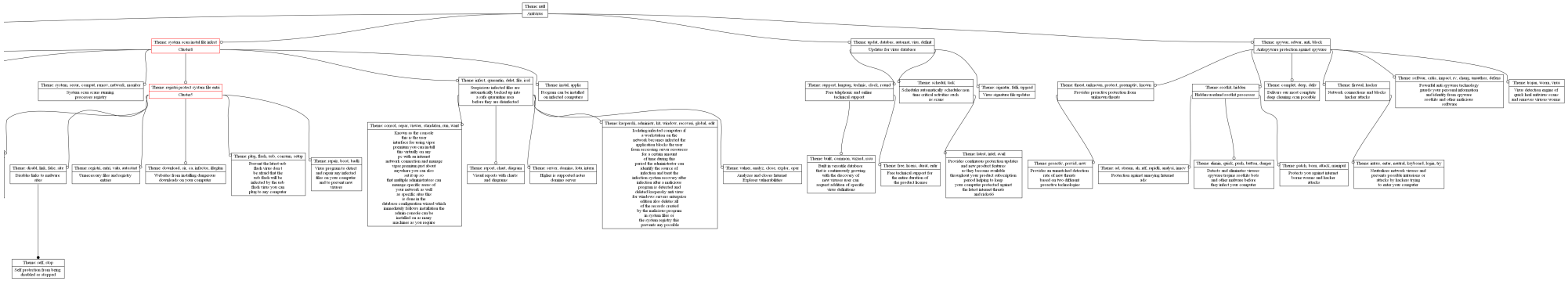


(b) (2/2)

Figure C.1: Feature Model extracted using only association rules



(a) (1/2)



(b) (2/2)

Figure C.2: Feature Model extracted using association rules and clustering